8-1-1989

A Shift variant filter applied to edge trace analysis

David C. Johnson

A SHIFT VARIANT FILTER APPLIED TO EDGE TRACE ANALYSIS

by

David C. Johnson

B.S. SUNY Fredonia

M.S. SUNY Geneseo

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in the Center for
Imaging Science in the College of
Graphic Arts and Photography of the
Rochester Institute of Technology.

August 1989

Signature of the Author **David C. Johnson**

Accepted by _____

Coordinator, M.S. Degree Program

COLLEGE OF GRAPHIC ARTS AND PHOTOGRAPHY
Rochester Institute of Technology
Rochester, New York

Certificate of Approval

M.S. DEGREE THESIS

The M.S. Degree Thesis of David C. Johnson
has been examined and approved by the thesis
committee as satisfactory for the thesis
requirement for the Master of Science degree.

Dr. Edward M. Granger, Thesis Advisor

Dr. Rodney Shaw

Dr. Ken Womack

5/3/89
Date

THESIS RELEASE PERMISSION FORM

ROCHESTER INSTITUTE OF TECHNOLOGY
COLLEGE OF GRAPHIC ARTS AND PHOTOGRAPHY

Title of Thesis: A SHIFT VARIANT FILTER APPLIED TO EDGE TRACE ANALYSIS

I, David C. Johnson, hereby grant permission to the Wallace Memorial Library of R.I.T. to reproduce this thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 8/3/89

A SHIFT VARIANT FILTER APPLIED TO EDGE TRACE ANALYSIS

by

David C. Johnson

Submitted to the
Center for Imaging Science
in partial fulfillment of the requirements
for the Master of Science Degree
at the Rochester Institute of Technology

ABSTRACT

A shift variant filter that automatically locates edge toe and shoulder points was used to reduce the noise in the plateau regions of simulated edges having uncorrelated noise. Edges were generated from Gaussian and triangle spread functions and represent edges produced by linear imaging systems only. Versions of these having flare were also used. For comparison a simple "chop" filter was applied to the same edges. Optical Transfer Functions were generated from the 12 categories of edges, ie. unfiltered, shift variant filtered, and chop filtered for each edge type. Modulation transfer functions of these were compared with MTFs of noiseless perfect edges for each category.

The results shows that the shift variant filter improves MTFs best for edges of medium to high noise having flare. When flare does not exist chop filter performance is superior. Surprisingly, no filtering at all results in the best MTF outputs in the case of low noise level edges with flare.

ACKNOWLEDGEMENTS

I wish to express appreciation to the following for their assistance in the successful completion of this research:

Dr. Edward Granger of the Eastman Kodak Co. and Adjunct Professor at the Rochester Institute of Technology, for acting as my thesis advisor, helping me select an area of research and helping me out of "jams".

Mr. Robert Chen, electrical research engineer at Videk, who helped me with the finer points of forth programming.

Dr. Ken Womack of Eastman Kodak Co. for being on my thesis committee.

Dr. Rodney Shaw, CIS Chairman, for being on my thesis committee.

DEDICATION

This thesis is dedicated to my wife, Jacqueline who helped make this possible by her support.

TABLE OF CONTENTS

List of Tables

List of Figures

1 Introduction

1.1 Imaging Systems

There is a need to have accurate information about how imaging systems affect final image output. With this information system problems can be diagnosed and design improvements made to yield better images. One form this information can take is the Modulation Transfer Function.

1.2 Modulation Transfer Function

The Modulation Transfer Function (MTF) is a measure of the ability of an imaging system to reproduce fine detail in the final image. It can be thought of as the ratio of output to input which is expressed as percent modulation for each spatial frequency (line pairs per millimeter). MTF is also the modulus of the complex valued Optical Transfer Function (OTF). The Optical Transfer Function can be obtained by finding the Fourier transform of the point spread function or the line spread function of the system. OTF is especially useful because the individual OTFs of each component in a linear system can be multiplied to obtain the system OTF. Photographic systems are not linear but if low contrast images are used, satisfactory approximations of image structure using MTF methods can be made.

1.3 Some methods for the Determination of the Modulation Transfer Function

1.3.1 A common method for measuring MTF is the sine-wave method. Sine-wave targets of known frequency and modulation are imaged by the system. The ratio of the output modulation to the input modulation then yields the modulation transfer function for each sine-wave used. Though reliable, the method is laborious and time consuming since for each frequency measured, a separate target must be used, data obtained, and then processed.

1.3.2 A faster method involves Imaging a line to yield the line spread function which when transformed leads to the Optical Transfer Function. However it is very difficult to produce and image a line that is fine enough. An alternative is to use an edge which is much easier to work with.

1.3.3 Analysis of edge information in an image can be used to find the OTF and is fast and convenient. Edges are not difficult to construct and are found naturally in most images. A microdensitometer is used to scan an edge in the image. The resulting data constitutes the edge function. This can be numerically differentiated to produce the line spread function.
The Optical transfer function can then be found by Fourier transforming the line spread function according to the equation below.

$$L(f) = i2\pi f \int_{-\infty}^{\infty} I(x) e^{-i2\pi fx} dx \qquad \text{Eq 1.1}$$

Where:

L(f) = optical transfer function

f = spatial frequency

I(x) = line spread function.

The real and imaginary parts of the optical transfer function are then used to generate the modulation transfer function:

$$L(f) = L(f)_r + L(f)_i \qquad \text{Eq. 1.2}$$

$$MTF = \sqrt{(L(f)_r)^2 + (L(f)_i)^2} \qquad \text{Eq. 1.3}$$

1.4 Dealing With Noise in Edge Trace Analysis

Deviations in data are greatly magnified during numerical differentiation of the edge function, therefore noisy edge data cannot be differentiated without smoothing the data in some way beforehand. Another alternative is to avoid differentiation altogether. Many methods for dealing with this noise problem have been used.

1.4.1 Several of the methods involve some technique for obtaining the MTF directly from the edge function without using numerical differentiation. Scott used square wave edge gradients on a hand smoothed edge function, (1). Barakat devised a mathematical method for getting the MTF directly from the edge but it did not take into account noise problems, (3). Cadou averaged 20 microdensitometer scans of the same edge with good results, (11). But this would not be possible if only one edge trace was available.

Schneiders used a direct technique that also required hand smoothing of the edge data beforehand. Hand smoothing has been shown to produce a reasonably accurate MTF but an inconsistent one, (9). There was a need to develop a way of dealing with edge noise that was consistent.

1.4.2 Overington and Brown used a Gaussian cumulative curve fitting technique that makes a significant step in the direction of smoothing noisy edge functions with consistent results, (8). A best cumulative normal curve is fitted to the edge data using least squares. Residual errors are then determined. The product of a third order polynomial with a Gaussian function is fitted to the residual error function. The sum of the cumulative Gaussian function and the fitted error function gives a good approximation of the noisy edge function which can then be differentiated to get the line spread function and subsequently the OTF.

The process is not without human judgment since a choice must be made in the selection of a standard deviation for residual error function curve fitting. One standard deviation brings about better fits in the toe and shoulder regions at the expense of the slope region whereas a different choice favors best fit in the slope region.

1.4.3 Jones reported on an automated method that provided treatment of the edge function before differentiation, (4). Smoothing was accomplished by convolution of this function, D(x) with the edge function:

$$D(x) = \frac{2kc}{X}\left(\cos(2\pi kcX) - \mathrm{sinc}(kcX)\right) \qquad \text{Eq. 1.4}$$

Where: **X** is the distance across the edge and **Kc** is the cut-off spatial frequency estimated from the edge shape. The resulting smoothed curve can then be differentiated to get the line spread function. The method was tested for its computational accuracy by using it on mathematically derived edges and comparing those MTFs with the ideal ones. The agreement was reported as being very good. In other tests of simulated edge motion there was some error but reasonable agreement with the ideal data. Again in this method there is dependence on a human judgment, ie. estimation of the spatial cut-off frequency.

1.4.4 Tatian (2) developed a way of estimating the OTF directly from a sampled edge using trigonometric series. It does not require differentiation so can be used on noisy edge data without smoothing. This method will be used in this study. The equations for the technique are shown below. A derivation of the equations can be found in appendix V.

The edge function can be divided into 3 sections. The lower plateau region ($x < N$), the slope region ($N < x < M$), and the upper plateau region ($x > M$).

Edge Function Terms

The lower plateau region is considered to be 0 so can be eliminated from the calculations. The slope region is evaluated using a summation from N to M. The upper plateau region is assumed to be normalized to 1 so can be represented by the right hand term in equations 1.5 and 1.6 on the next page.

$$L_R(f) = 2\pi f\varepsilon \sum_{n=N}^{M} e(x) \sin(-2\pi f n\varepsilon) + \frac{\cos\left(\left(M + 1/2\right)\right)2\pi f\varepsilon}{\mathrm{sinc}(f\varepsilon)}$$

Eq 1.5

$$L_R(f) = 2\pi f\varepsilon \sum_{n=N}^{M} e(x) \cos(-2\pi f n\varepsilon) - \frac{\sin\left(\left(M + 1/2\right)\right)2\pi f\varepsilon}{\mathrm{sinc}(f\varepsilon)}$$

Eq 1.6

Where:

$L_R(f)$ and $L_I(f)$ are the real and imaginary parts of the optical transfer function, respectively,

f is the spatial frequency,

e(x) is the edge function,

N is the x position of the beginning of edge data being considered

M is the x position of the end of edge data being considered

and $\mathrm{sinc}(u) = \sin(2\pi u)/2\pi u$.

1.5 Characteristics of an Ideal OTF Method

Having surveyed techniques for finding OTFs from noisy edge traces it will be useful here to list some characteristics which an improved method should have.

* Make use of a single edge or trace of an edge (no averaging)

* Use a transformation that can tolerate some noise (no numerical differentiation)

* Use discrete data

* Avoid smoothing (valuable edge information can be lost),

* Avoid subjective judgments involving parameter selection,

* Return an accurate OTF .

2 Methods

This research was done entirely with computer simulations. Noise was added to edges generated from spread functions. Edges were treated in various ways including the shift variant filter and used to generate MTFs which were stored in files. Effectiveness of the shift variant filter was measured by analyzing the MTF data.

2.1 Edges

Each edge was generated by first creating a spread function and then integrating it. Two types of spread functions were used, Tri(x/b) and Gauss(x/b) where:

$$\mathrm{Tri}\left(\frac{x}{b}\right) = \begin{cases} 0, & \left|\frac{x}{b}\right| \geq 1 \\ 1 - \left|\frac{x}{b}\right|, & \left|\frac{x}{b}\right| < 1 \end{cases} \qquad \text{Eq. 2.1}$$

and

$$\mathrm{Gauss}\left(\frac{x}{b}\right) = e^{-\pi \left(\frac{x}{b}\right)^2} \qquad \text{Eq 2.2}$$

All spread functions and resulting edges are centered on a 256 point continuum. For the triangle function b is 30 resulting in a triangle of base 60. For the Gaussian function b is normally 1. In order to add flare a Gaussian spread function was created where b = 4 which has the effect of flattening and extending the curve.

Spread Functions with flare were constructed using the following relationships:

$$\textbf{Gauss with Flare} = .9\ \textbf{Gauss}\left(\frac{x}{1}\right) + .1\ \textbf{Gauss}\left(\frac{x}{4}\right) \qquad \textbf{Eq } 2.3$$

$$\textbf{Triangle with Flare} = .9\ \textbf{Tri}\left(\frac{x}{30}\right) + .1\ \textbf{Gauss}\left(\frac{x}{4}\right) \qquad \textbf{Eq } 2.4$$

Examples of these spread functions and their resulting edges can be found in figures 2.1 to 2.5 on the following pages.

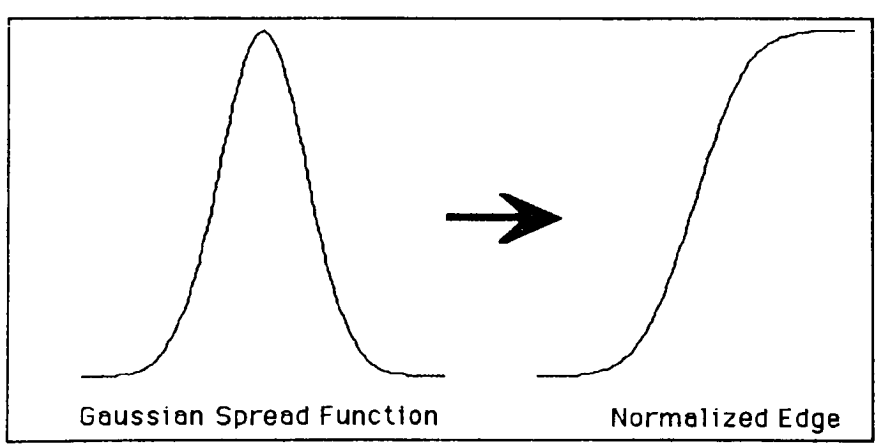


Figure 2.1 Gaussian Spread function and resulting edge.

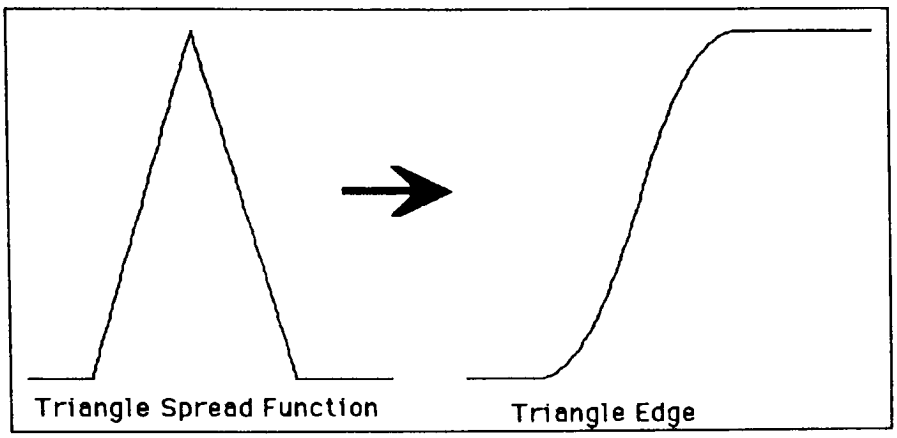


Figure 2.2 Triangle Spread function and resulting edge.

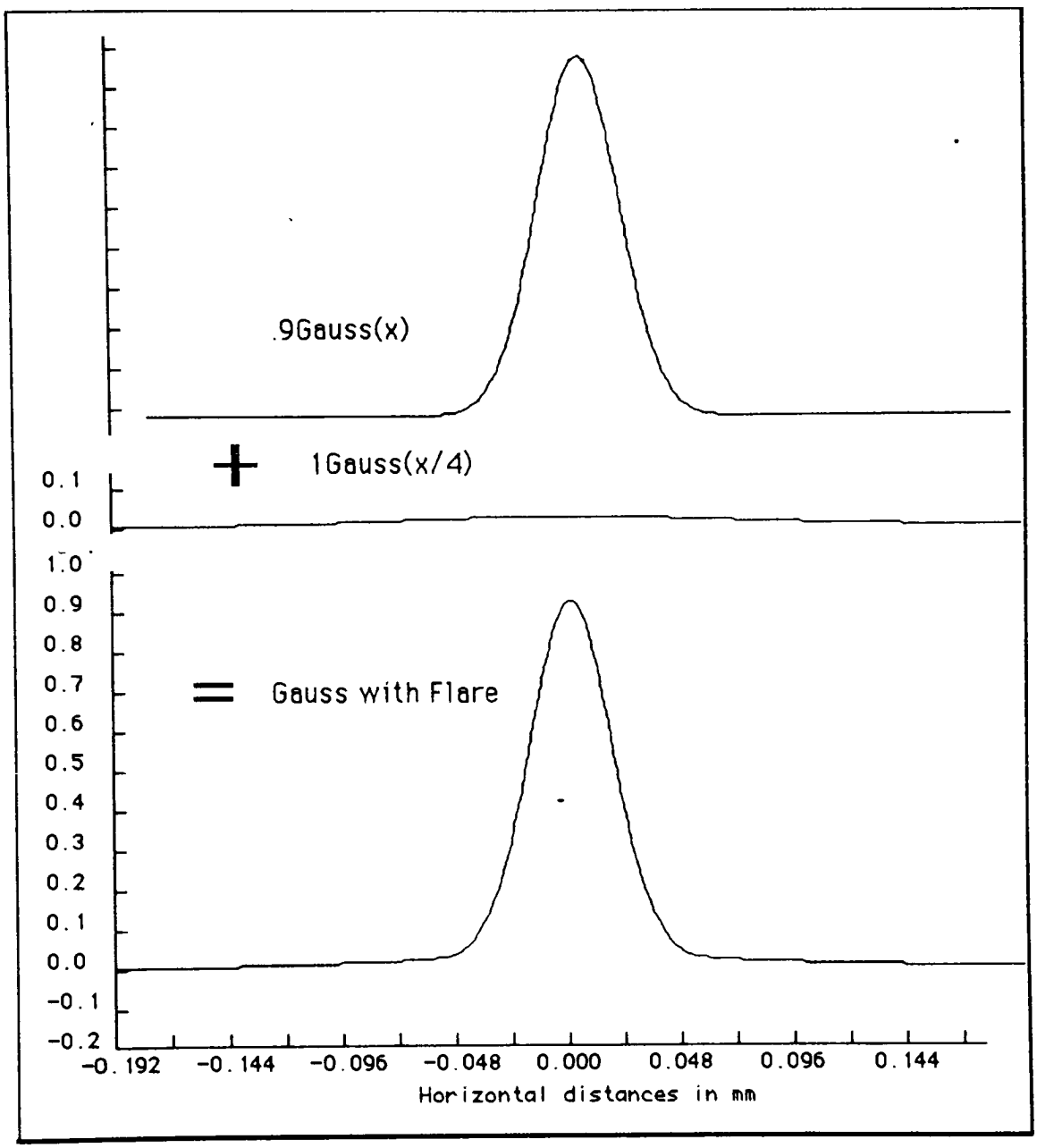


Figure 2.3 Flare Spread function showing addition of 2 Gaussian functions

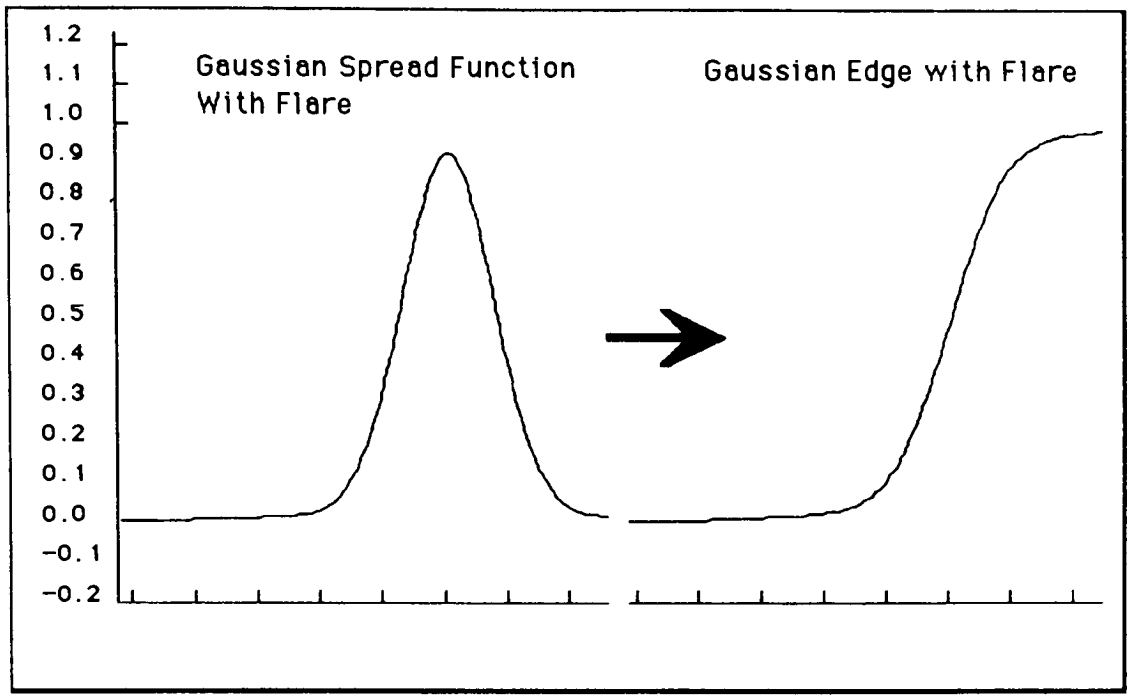


Figure 2.4 Gaussian spread function with flare and resulting edge.

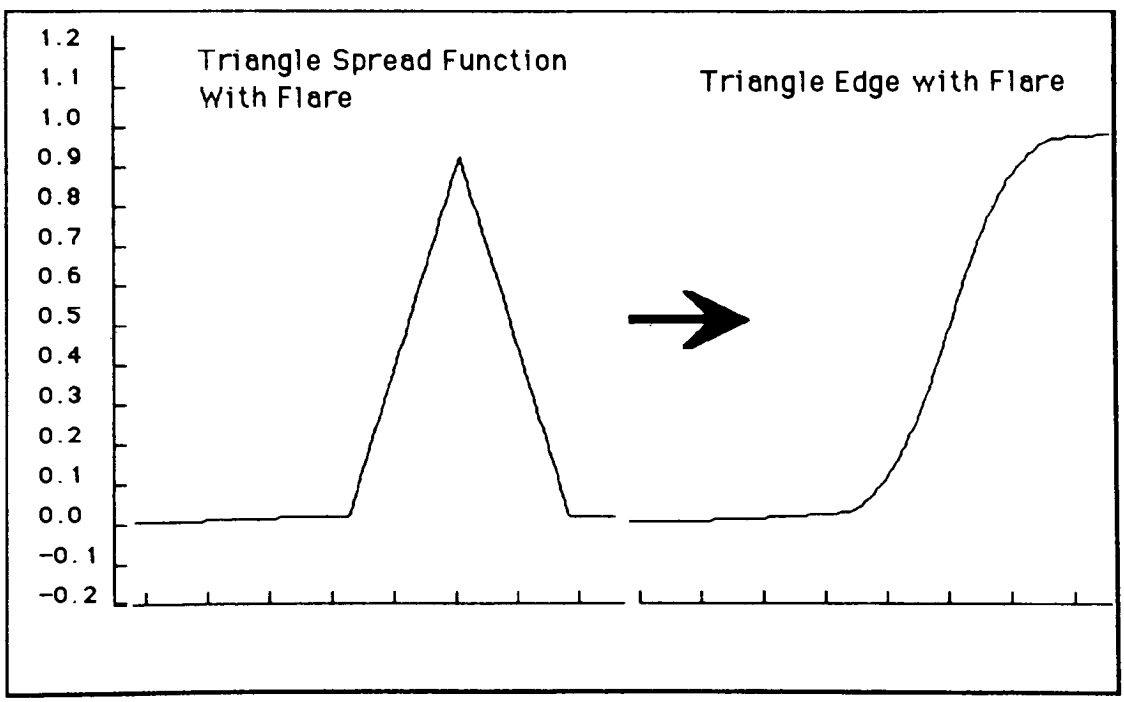


Figure 2.5 Triangle with flare spread function and resulting edge.

2.2 Noise

Normally distributed uncorrelated noise was added to each edge point by point. 100% RMS noise was generated using the expression:

$$n(\varphi, 1) = \left[\sum_{1}^{12} U(\varphi, 1)\right] - 6 \qquad \text{Eq. 2.5}$$

Where :

$n(\varphi, 1)$ is the 256 point noise function and

$U(\varphi, 1)$ represents 256 random numbers between 0.000 and 1.000.

Edges having noise RMS levels of 1%, 3% and 10 % were achieved by multiplying each $n(\varphi, 1)$ by the appropriate fraction. Examples of these edges follow.

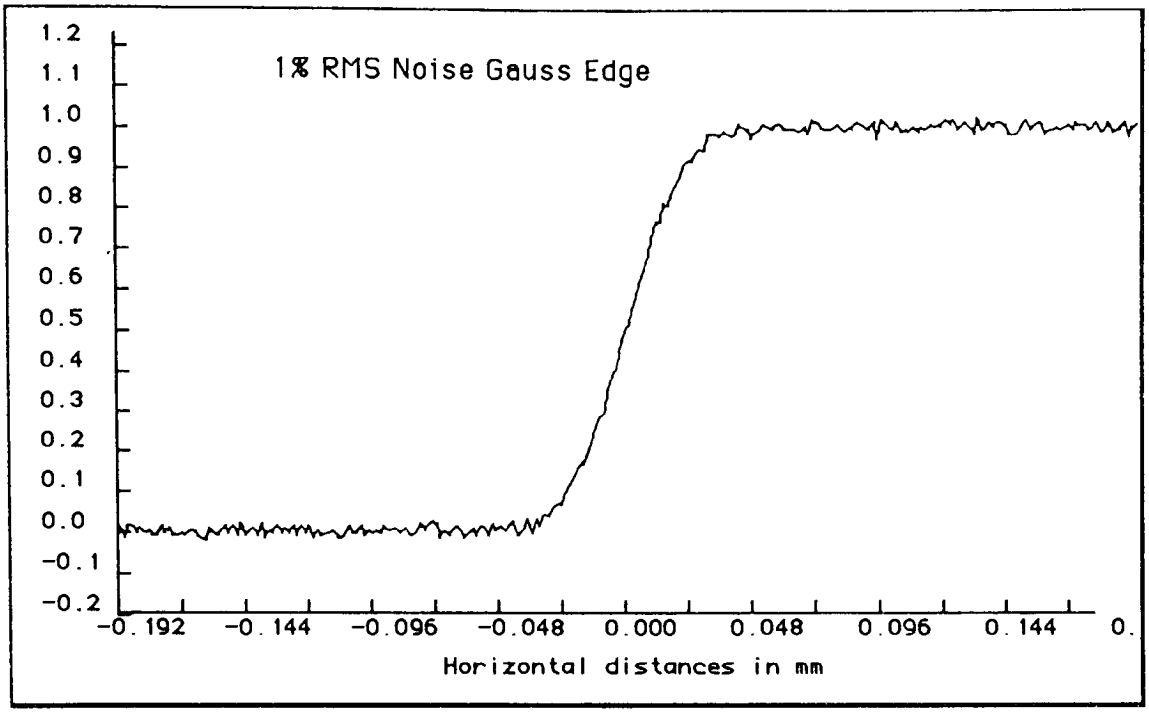


Fig 2.6 Gaussian edge with 1% noise.

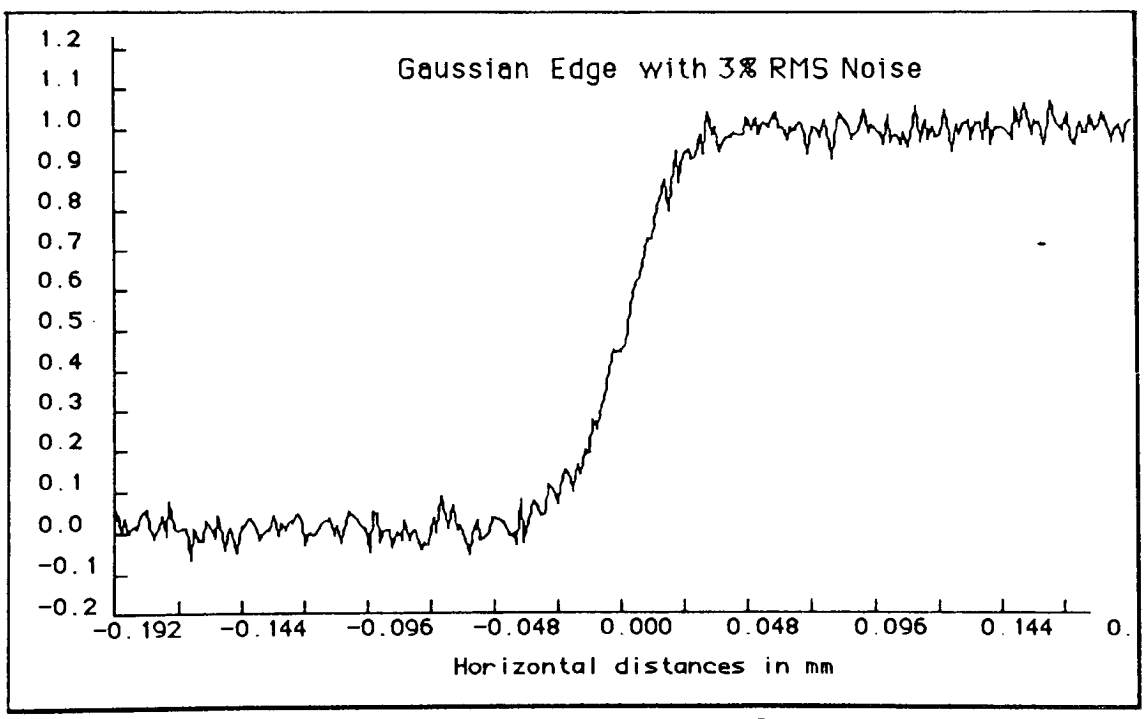


Figure 2.7 Gaussian edge with 3% RMS noise.

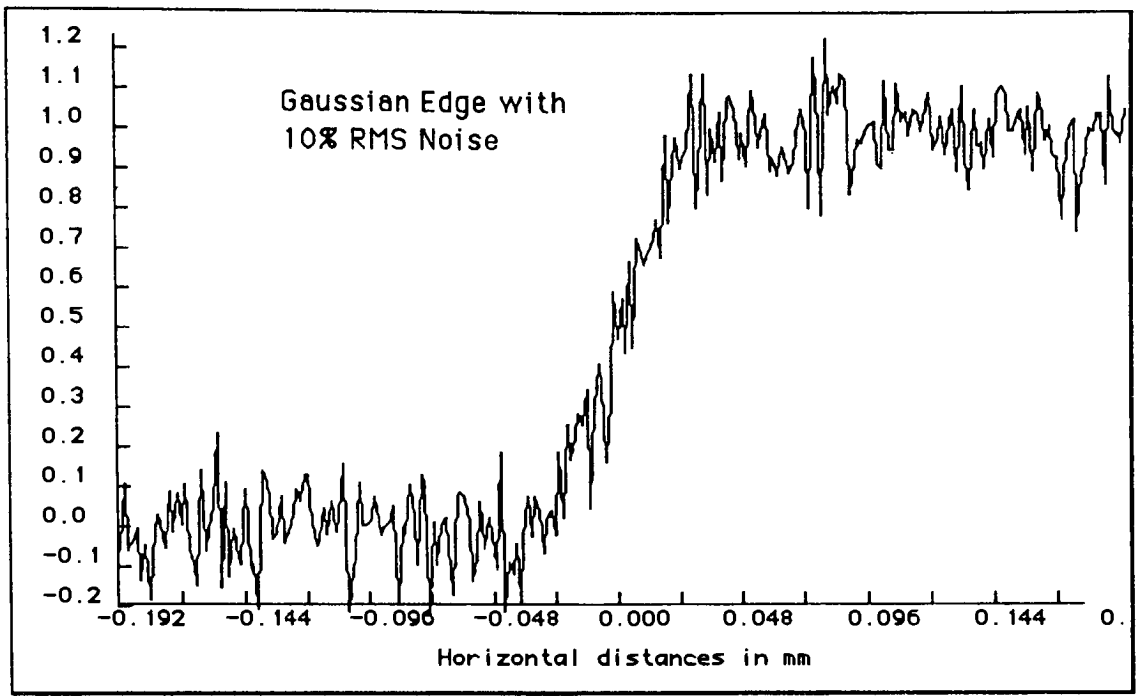


Figure 2.8 Gaussian edge with 10% RMS noise.

2.3 Filters

The strategy followed in treating the noise in the edges was to eliminate as much noise as possible in the plateau regions without altering any information in the gradient portion of the edge. The filter was also intended to function automatically on generic edge data without having to set any special parameters based on a foreknowledge of the edge. This was achieved except perhaps in one instance which will be discussed later.

2.3.1 Toe /Shoulder Location

An important first step was to accurately estimate the location of the boundary between the low plateau and the gradient and between the high plateau and the gradient. These two points will hereafter be referred to as the toe and shoulder of the edge respectively. Four different methods were tried before the Arc Sine Method was decided upon as the best boundary estimator. Most of the following discussion will be about that method although the others will be briefly described.

2.3.1.1 Block averaging

All four methods begin by finding the plateau averages and then using these to normalize the edge data. The entire edge is then smoothed using block averaging. It should be pointed out here that this smoothed edge is only used to locate the toe and shoulder points. Once these points are located the filter is applied to the original noisy edge data.

2.3.1.2 Arc Sine Method

For the arc sine method the center of the edge is first located. This is done by finding the x values of two matched points on the smoothed curve, one n% above the low plateau and the other n% below the high plateau. The midpoint between these was taken as an estimate of the edge center. To get a more accurate estimate this was carried out a total of 10 times using 10 different matched points and then averaging. n ranged between 15 and 40.

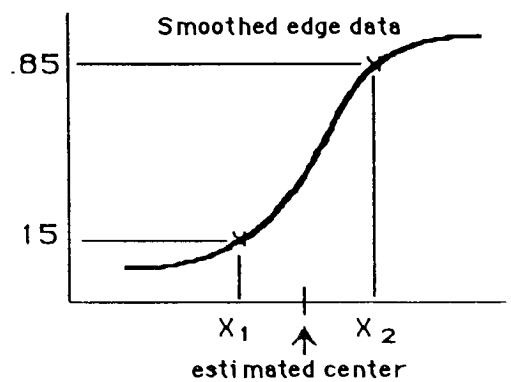


Fig. 2.9 Matched point method for estimating center

Between the toe and shoulder points the edge function can be approximated as the sine function:

$$\left[E(x) = \frac{1}{2} \mathrm{Sin}\left(\frac{x}{k} \pi \right) + .5 \right]_{\frac{x}{k} - \frac{1}{2}}^{\frac{x}{k} - \frac{1}{2}} \qquad \text{Eq. 2.6}$$

where E(x) is the edge function and k is a scaling factor. For a particular edge there will be one value of k that defines the best sine function fit. Solving for K in equation 2.6 obtain:

$$K = \frac{x \pi}{\sin^{-1}(2E(x) - .5)} \qquad \text{Eq. 2.7}$$

Equation 2.7 was used to find values of K for every x, E(x) coordinate pair from E(x) = 0.1 to E(x) = 0.9 . The average of these values of K was taken to be the scaling factor for the matching sine function. The toe and shoulder points were then found by solving for x in equation 2.7 with the estimated k value in place. The toe point was found by substituting 0 for E(x) and the shoulder point by substituting 1 for E(x).

2.3.1.3 Smoothing Displacement Compensation

The toe and shoulder thus obtained are the best estimated points for the smoothed data but not the real edge data. When the edge data was block averaged the apparent toe and shoulder points were displaced out from the center by an amount proportional to the size of the

averaging block. This offset was measured by running the arc sine algorithm on a noiseless Gaussian edge where the toe and shoulder points were known. In the final algorithm the offset was subtracted from the estimated toe point and added to the estimated shoulder point.

2.3.1.4 Other Toe/Shoulder Location methods

Three methods that were tried involved using thresholds. Smoothed data was evaluated in serial order and If a preset threshold was exceeded a sufficient number of times the toe point was established. Reversing the process established the shoulder point. Offsets were also used for final adjustment. For the three methods the data being considered was 1) smoothed curve data, 2) first derivative of data or slope of the smoothed curve, and 3) second derivative of the data or change of slope of the smoothed curve. Thresholds were also automatically adjusted according to the measured standard deviation of the plateau data. Curve fitting to a third order polynomial was also tried. All of these methods were judged to be less reliable in comparison to the arc sine method.

2.3.2 **Shift Variant Filter** (damping Filter)

The equation for the shift variant edge filter function is given below:

$$E(m) = \frac{\sum_{I=n}^{(2Q-1+n)} e(i)}{(2Q-1)} \qquad \textbf{Eq. 2.8}$$

Where:

E(m) is the filtered plateau function,

Q = M - N (distance from the toe/shoulder),

M is the toe/shoulder point,

n is the current x axis value of the edge function,

e(i) is the original noisy edge function,

and **2Q - 1** is the number of points being averaged.

The filter starts at the toe or shoulder point and works out away from the slope portion of the edge. It can be thought of as block averaging with the block starting out very small (1 unit) and increasing in size as the filter moves farther from the toe/shoulder point. In this way the filter accounts for gradual trends in the data of the plateaus while eliminating most of the noise.

2.3.3 Chop Filter

An alternative method of filtering is to simply chop off the noise in the plateau regions then assign 0 to all low plateau points and 1.0 to all high plateau points. Once the toe and shoulder have been located this is easily accomplished. It was decided to include this "chop" filter to serve as a second experimental control in evaluating the effectiveness of the shift variant filter. Examples of filtered edges are shown below.

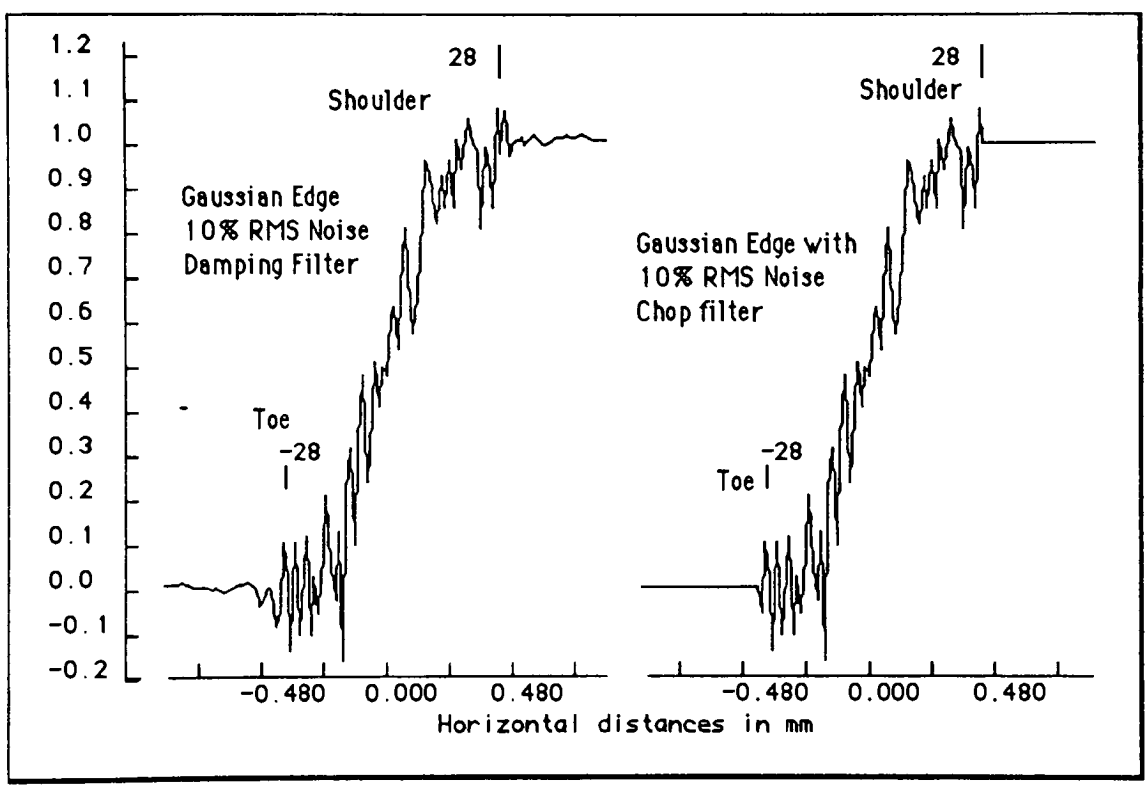


Figure 2.10 Two treatments of the same noisy edge. Damping filter on the left and chop filter on the right. Toe and shoulder points are marked as estimated by the Arc Sine algorithm.

2.4 Experiments

2.4.1 Test Edges

With the random noise generator 20 different noise edges were generated for each of the four categories of spread functions and stored in files. These were then used as the common data for all experiments. Edges for each category with no noise were also created to be used as standards in evaluating filter effectiveness.

2.4.2 Edge Treatment

Each noisy edge was given three different treatments and then transformed using Tatian's method. The three treatments were:

1) No filtering,

2) Damping filtering,

3) Chop filtering.

The table below summarizes data preparation for the filter evaluation experiment.

DATA SUMMARY FOR FILTER EVALUATION EXPERIMENT

Spread Functions	Noise Edges	Number of Filter Treatments			Total MTF Files
		No Filter	Damping	Chop	
Gaussian					
	1%	20	20	20	60
	3%	20	20	20	60
	10%	20	20	20	60
Gaussian With Flare					
	1%	20	20	20	60
	3%	20	20	20	60
	10%	20	20	20	60
Triangle					
	1%	20	20	20	60
	3%	20	20	20	60
	10%	20	20	20	60
Triangle With Flare					
	1%	20	20	20	60
	3%	20	20	20	60
	10%	20	20	20	60
Totals		240	240	240	720

Table 2.1 Data file categories and quantities.

2.4.3 Evaluation

Each of the 20 MTF functions for a particular level of noise and category of spread function was compared with a zero-noise standard MTF for that spread function category. This was done in two ways.

First, the root mean square differences (RMSD) for the 20 edges and the standard MTF was calculated using the equation:

$$RMSD = \frac{\sum_{n=1}^{n=20} \left[\frac{\sum_{m=1}^{m=I} \sqrt{[L(m,n) \quad S(m)]^2}}{i} \right]}{20} \qquad \text{Eq. 2.9}$$

Where:

RMS D is the root mean square, differences,

L(m,n) represents the 20 MTF functions,

S(m) represents a standard MTF function and

I is the number of points in the MTF function being considered. Below a modulation level of 20% values can be erratic. The cutoff point was determined by selecting the frequency where the standard MTF modulation dropped to 20%.

The resulting RMS differences for all the experiments can be found in a table in article 3.1 on page 27. The RMS differences number is an index of the overall match of 20 MTFs with the standard MTF. It is given in % modulation units. 0.0 % is a perfect match. 1.0% is a very poor match since the RMS differences number is the mean error for each point in the MTF

function. If the mean is 1.0% then the cumulative error for the function is n x 1% modulation (n being the number of points used). 99% intervals of confidence were also determined using the statistical "T" test. These are included in the tables and shown graphically in section 3.

RMSD is useful for making a quick evaluation but gives no information about modulation errors at specific frequencies. That is why a second method that finds the average absolute difference for each frequency of the MTF function was devised. The equation for the average absolute difference function , **D(m)** for a block of 20 MTFs is given below:

$$D(m) = \frac{\sum_{n=1}^{n=20} \left|\left[L(m,n) - S(m)\right]\right|}{20} \qquad \text{Eq. 2.10}$$

A composite plot of the difference functions of the three filter treatments can reveal interesting detail about the effects of filtering on MTF output. A one fifth scale standard MTF curve was included to aid in interpretation. These plots can be found in the results section on pages 33 to 34 and in appendix II , pages 51 to 59. A similar plot of the difference functions for the 3 noise levels of a particular edge treatment is a useful tool in assessing noise tolerance and is included in the results section.

A third assessment tool was used that enables a visual evaluation of edge filtering effects on a set of 20 edges. This is a composite plot of 20 MTFs on a frequency log scale. A standard MTF of a noiseless edge is also included in each plot as a dashed line. These graphs can be found in the discussion on page 36, in Appendix II pages 50 to 59, and Appendix III pages 60 to 75.

3 Results

3.1 MTFs of Edges with Gaussian and Gaussian with Flare Spread Functions

Below is a table of RMSD data for Gaussian and Gaussian with flare edge MTFs. The RMSD is a mean error for the 20 MTF samples. It is possible to predict the RMSD value for the entire population of possible MTFs of a specific edge noise level and spread function type. The output of the T test is a 99% confidence interval of modulation differences. This can be done by applying the statistical " Student's T Test" for small samples. (14)

$$\text{Interval of Conf.} = \bar{X} \pm t \frac{s}{\sqrt{n}} \qquad \text{Eq. 3.1}$$

Where:

$\bar{X}$ is the sample mean

t is the "T test" number for 19 degrees of freedom obtained from a table for 99% confidence intervals

s is the sample standard deviation

n is the number of samples.

This information is more easily interpreted when presented as a chart (see pgs. 29 to 31). Each vertical line between the diamonds in the chart represents the range of modulation differences that would be expected to occur with a probability of 99% for a particular population of MTFs. The square mark indicates the position of the mean. Each line can be thought of as the base of an equal area normal curve with truncated tails. The area under the curve is proportional to probability. If the line is short the curve is taller which means

there is a greater probability of an actual modulation difference occurring at or near the mean. The probability of a modulation difference occurring near the end of a line is very low and above or below the line is less than 1% . With that in mind the chart will be used to help interpret the data.

ROOT MEAN SQUARE MODULATION DIFFERENCES
COMPARING MTFS OF NOISY EDGES WITH NOISE FREE EDGE MTFS
GAUSSIAN & GAUSSIAN WITH FLARE

Edge Type Spread Function & RMS Noise	Filter	From 20 to 100% Modulation Mean %	Standard Dev. %	99% Confidence Interval Limits
Gaussian				
1%	None	.103	.040	.077 to 129
1%	Damping	.061	.037	.045 to .077
1%	Chop	.057	.033	.037 to .077
3%	None	.255	.155	.181 to .329
3%	Damping	.205	.131	.135 to .275
3%	Chop	.175	.092	.105 to .245
10%	None	1.058	.408	.799 to 1.317
10%	Damping	.796	.277	.575 to 1.017
10%	Chop	.568	.278	.342 to .794
Gauss + Flare				
1%	None	0.094	0.040	.068 to .120
1%	Damping	0.102	0.037	.078 to .126
1%	Chop	0.344	0.033	.322 to .366
3%	None	0.322	0.155	.220 to 424
3%	Damping	0.246	0.131	160 to .332
3%	Chop	0.387	0.092	.327 to .447
10%	None	0.956	0.408	.688 to 1.224
10%	Damping	0.627	0.277	.445 to .809
10%	Chop	0.656	0.278	.474 to .838

Table 3.1 Comparisons of test MTF data with standard noiseless MTFs. Gaussian and Gaussian with flare type edges.

3.1.1 Effect of Edge Noise on MTF

It can be seen that overall error in the MTF is directly proportional to the level of noise in the edge in two ways. First, the value of the RMSD (mean error) and second, the extent of the 99% confidence interval of the mean error. This latter effect can be explained by the fact that randomness in the noise of each edge results in random degrees of overall distortion between edges so that some distorted edges result in MTFs with greater relative errors than others. The only exception to the proportionality principal is at the 3% noise level of edges with flare. Both the mean error and confidence interval are greater than might be expected.

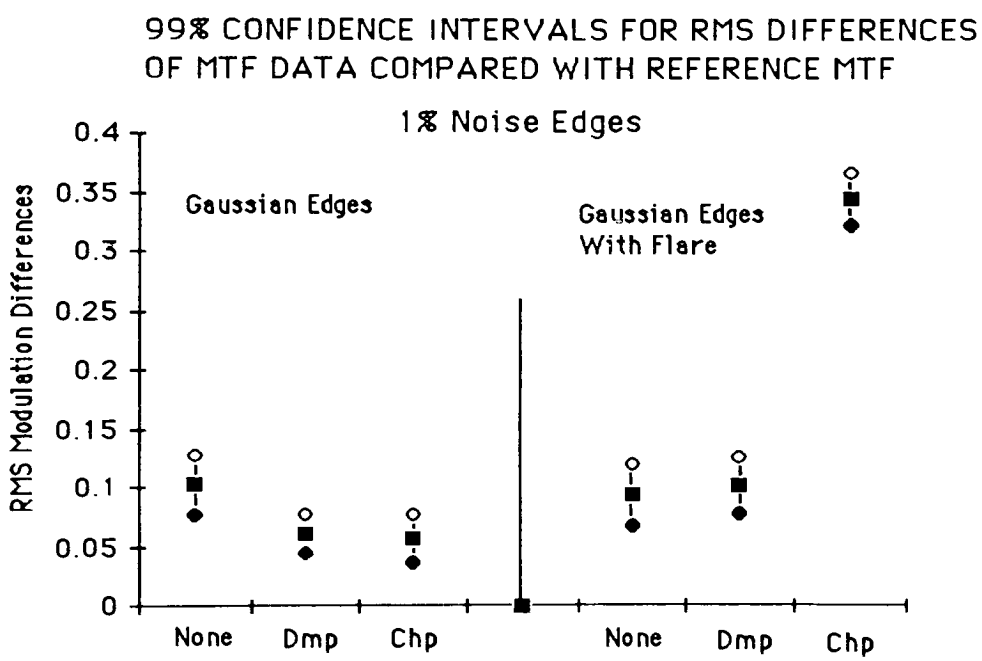


Figure 3.1 Chart showing RMSD for MTFs derived from Gaussian and Gaussian with flare 1% noise edges. 3 treatments for each edge type are: no filter (None), damping filter (Dmp), and chop filter (Chp). The white and black diamonds delineate the 99% confidence interval with the black square indicating the mean.

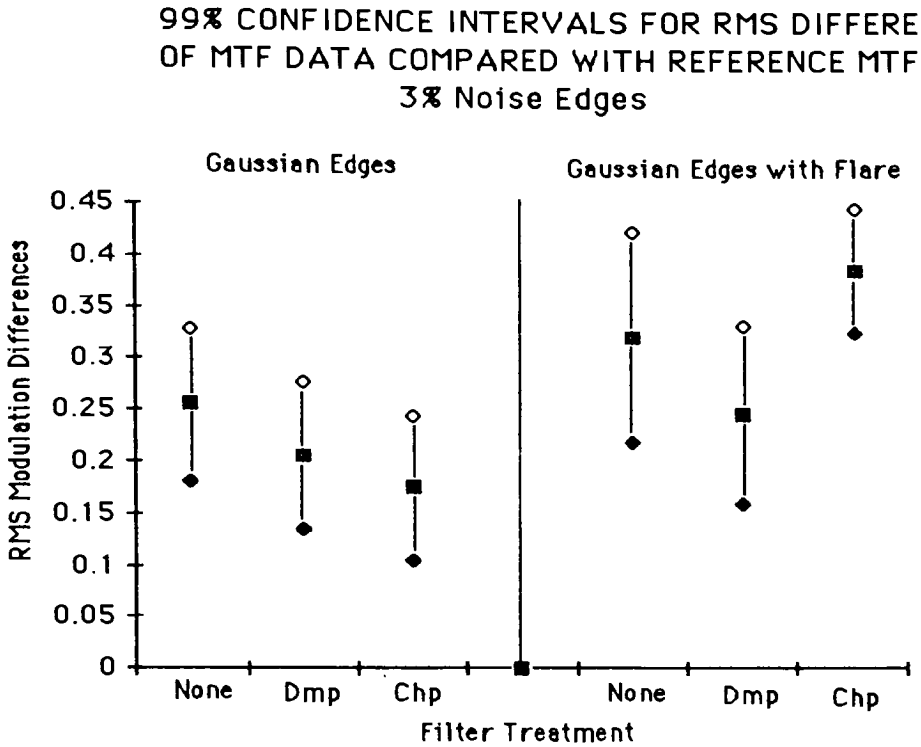


Figure 3.2 Chart showing RMSD for MTFs derived from Gaussian and Gaussian with flare 3% noise edges. 3 treatments for each edge type are: no filter (None), damping filter (Dmp), and chop filter (Chp). The white and black diamonds delineate the 99% confidence interval with the black square indicating the mean.

3.1.2 Effect of Filters on MTF

In general it can be said that both filters usually produce significant improvements in error means and confidence levels. One exception occurs at the 1% noise level for Gaussian edges with flare. In this case the unfiltered edge results in an MTF with less error than either filtered edge. Evidently the filters introduce more distortion than the low level noise that was removed. This phenomena will be discussed in more detail in the next section.

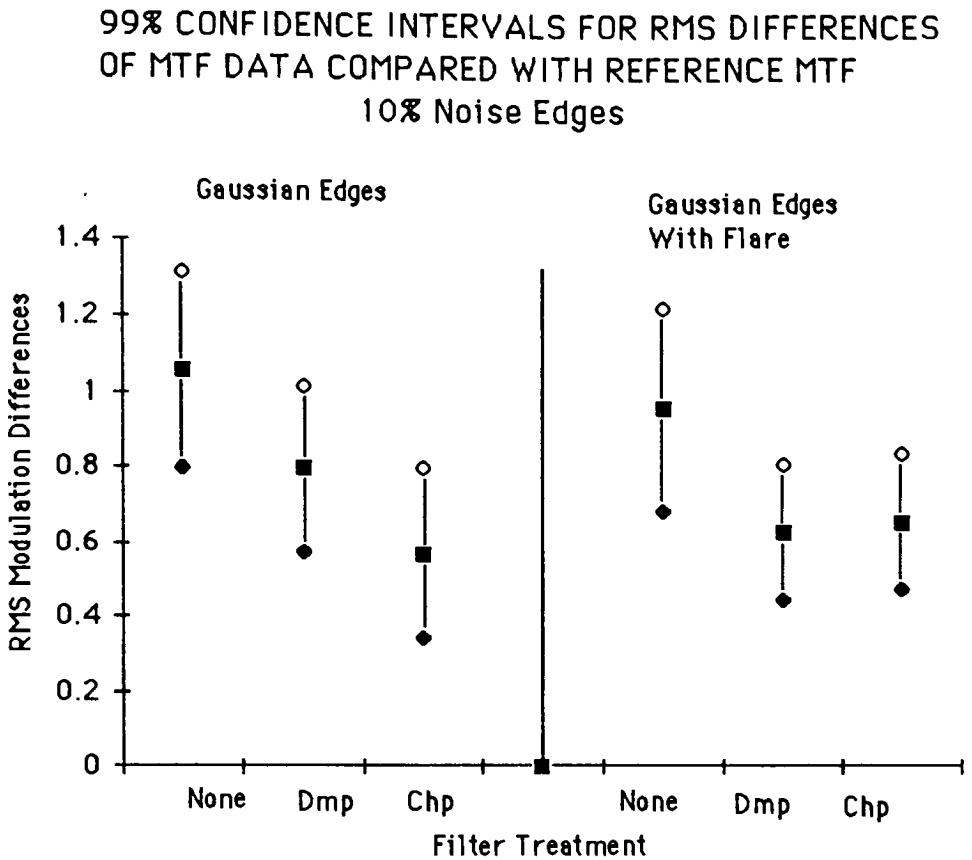


Figure 3.3 Chart showing RMSD for MTFs derived from Gaussian and Gaussian with flare 10% noise edges. 3 treatments for each edge type are: no filter (None), damping filter (Dmp), and chop filter (Chp). The diamonds delineate the 99% confidence interval with the black square indicating the mean.

For Gaussian edges without flare the chop filter outperforms the damping filter in improvement of error mean in every case. At the 10% noise level the chop filter results in a 46% improvement in mean error as contrasted to the damping filter improvement of 25%. At the 1% noise level the improvements are less disparate with chop filter at 45% and damping filter at 41%. However the damping filter outperforms the chop filter in improving the 99%

confidence interval: 35%, damping to 20%, chop at the 1% noise level and 6%, damping to 5%, chop at the 3% level. From the charts it can be seen that mean error is the more significant improvement of the two so the chop filter can be regarded as the best one in dealing with noise in Gaussian edges without flare. When flare is present in the edge the chop filter fails totally at the 1% and 3% noise levels and is inferior to the damping filter at the 10% noise level. Reasons for this will be presented in the next section on spatial frequency response.

Similar results were obtained for triangle and triangle with flare spread function edges. The data and charts for these can be found in the appendix on pages 60 to 75. One difference occurred for 10% noise triangle edges with flare. At this noise level the chop filter produced a 22% improvement in error mean contrasted to the 17% improvement brought about by the damping filter. The damping filter improved the 99% confidence interval by 11% and the chop filter made it worse by 1%.

Finally one issue needs to be discussed before leaving the topic of RMS differences. The 99% confidence intervals overlap at times, for instance in the case of filtered vs non filtered edges at the 10% noise level. One interpretation of this is that it would be possible for a nonfiltered edge to have a more accurate MTF than the same edge after filtering even though the probability for this would be low. However this is not the case. In the 20 edges that formed the sampling set for a particular noise level there is a 1 to 1 correspondence between edges before and after filtering. Where the RMSD data indicates it there will always be an improvement in the filtered edge MTF over its unfiltered counterpart MTF.

3.1.3 Effects of Noise and Filtering over the MTF Spatial Frequency Band

Average absolute error data for each frequency point is presented in the graphs below. A 1/5th scale reference MTF has been included in each. Plots for the 3 edge treatments are superimposed on one graph. The .20 modulation frequency limit point as used in calculating RMS differences is also shown. Below a frequency of 5 the damping filter has no beneficial effect. The chop filter appears to improve the MTF at all frequencies shown.

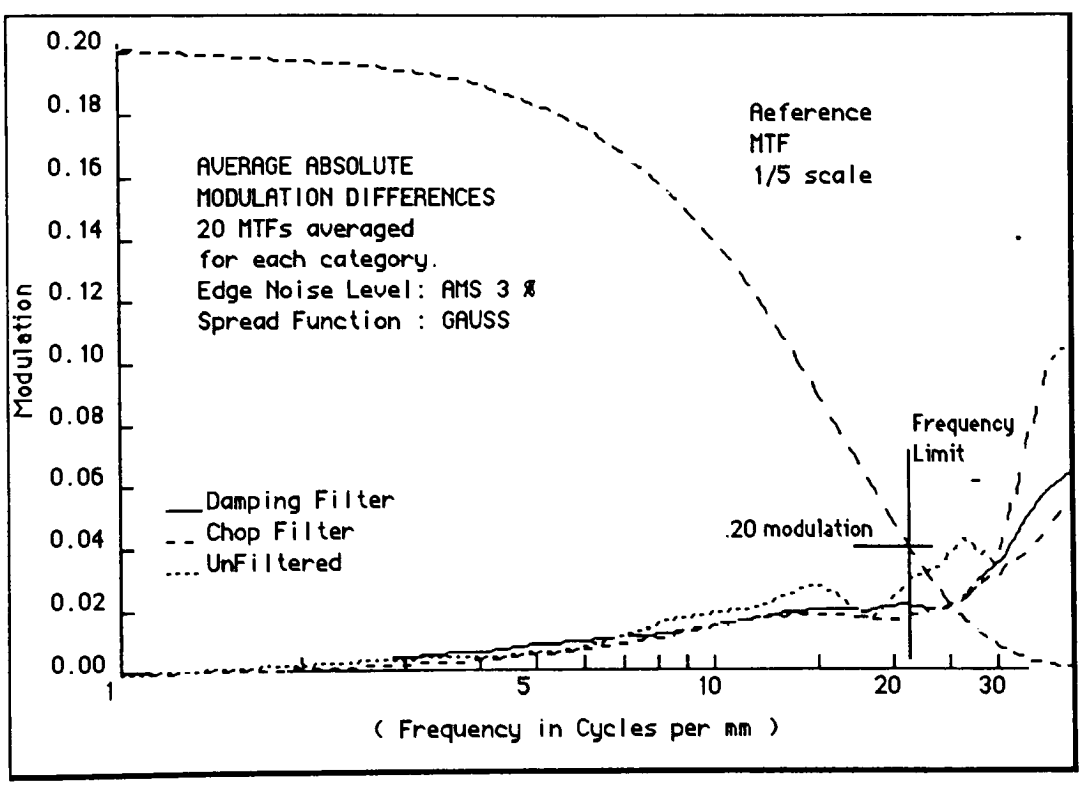


Figure 3.4 Average absolute Modulation differences by frequency for 3% Gaussian Edges.

For Gaussian edges with flare neither filter provides any improvement in MTF below 8 Cpmm. From that point to the frequency limit the damping filter results in a significant MTF improvement. Note the erratic nature of the chop filter plot. This can be explained by examining the 20 MTF composite plots on the next pages. The reference MTF is shown as a dashed line which is only partly visible. For the Gaussian edge without flare it can be seen that the 20 MTFs of chop filtered edges are distributed somewhat equally on both sides of the reference MTF.

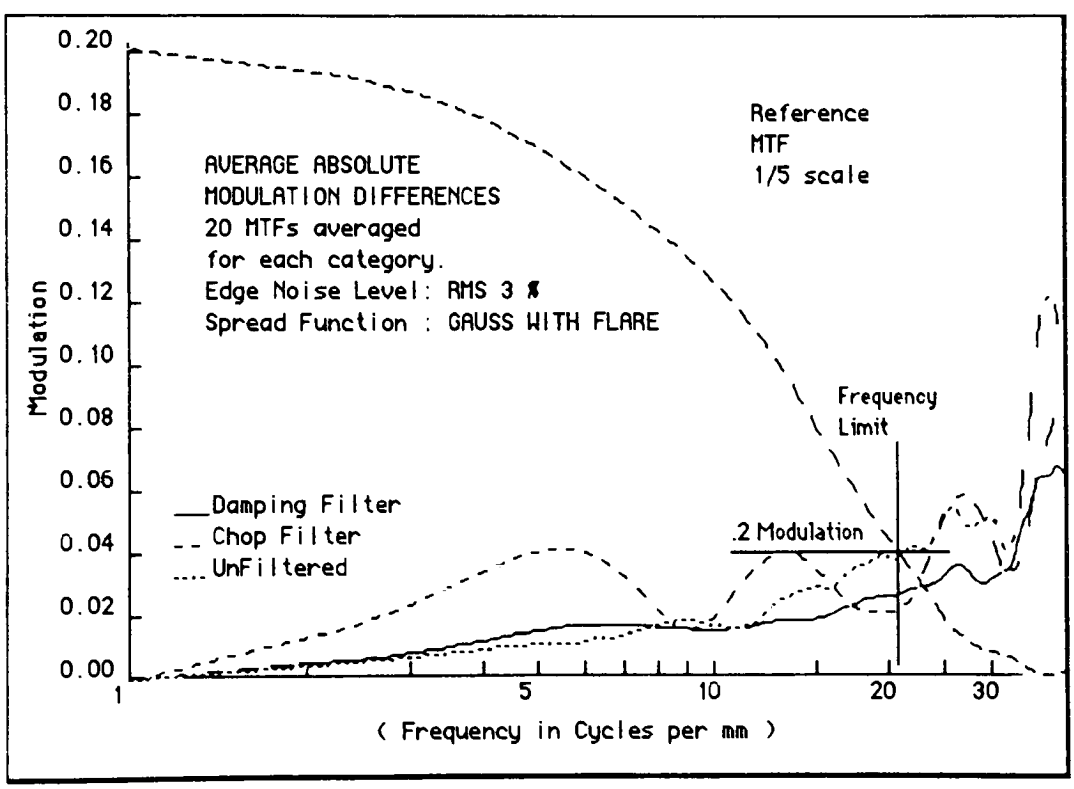


Figure 3.5 Average absolute Modulation differences by frequency for 3% Gaussian Edges with flare.

However for the Gaussian edge with flare all 20 MTF functions are biased above the reference MTF below 9 Cpmm and biased below the reference at frequencies above 9 Cpmm. This distortion is evidently caused by the sharp transition produced by the chop filter at the toe and shoulder points of edges with flare. A 3% noisy edge with flare is shown below with both filters applied. Since the damping filter averages in the flare effect much less distortion results than with the chop filter.

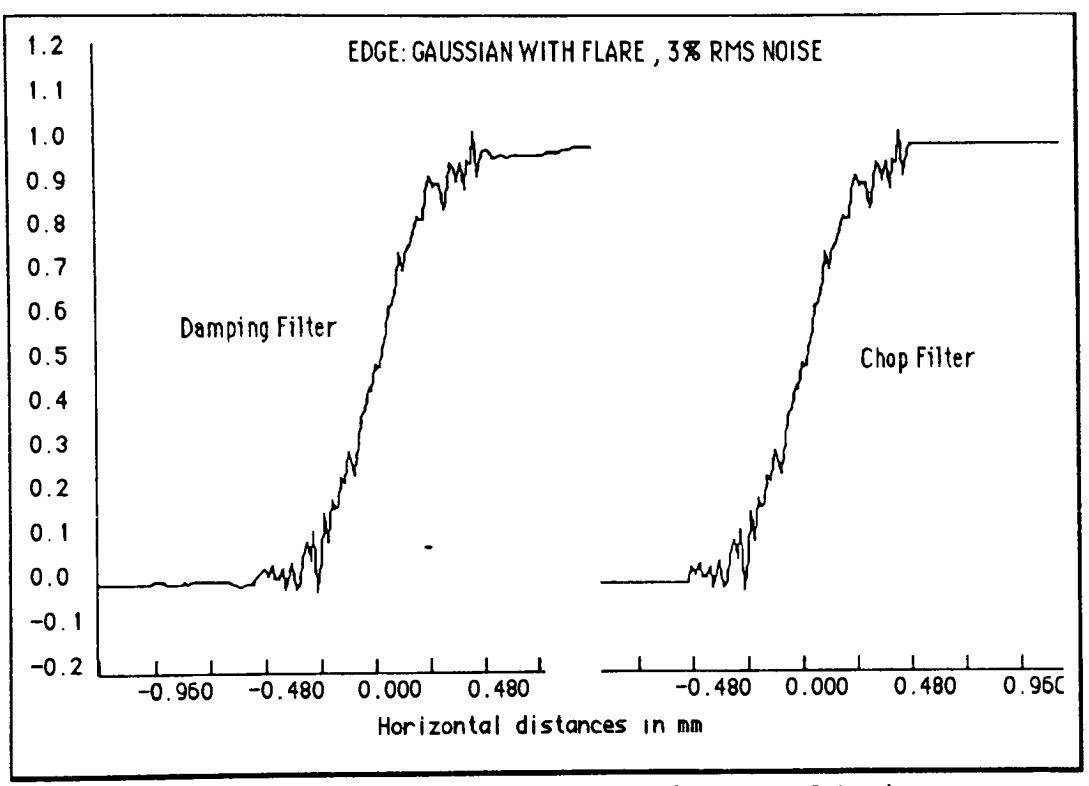


Figure 3.6 Damping filter and chop filter treatments of the same 3% noise Gaussian edge with flare present.

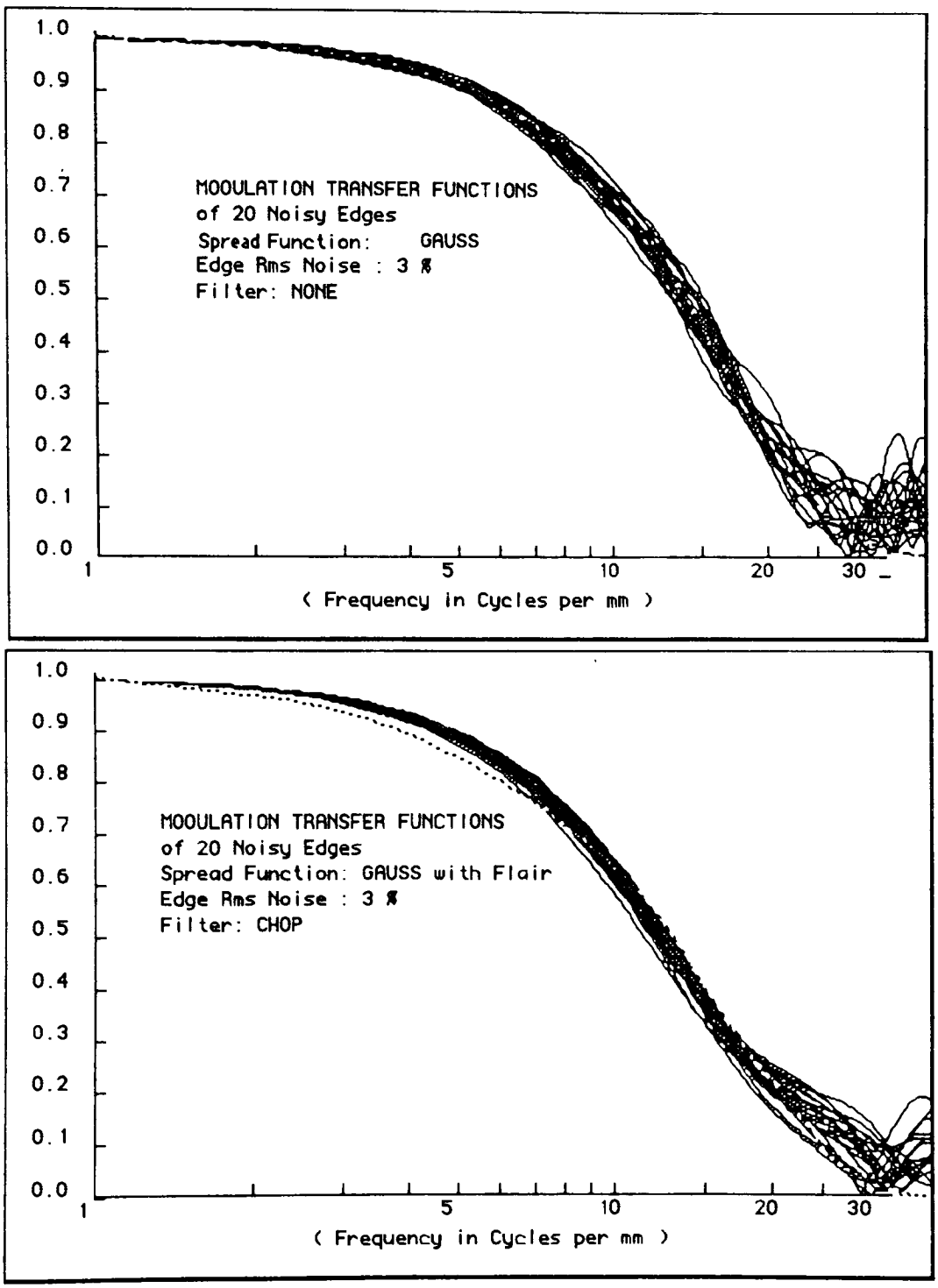


Figure 3.7 Composite graph of the 20 MTF plots of 3% noise edges used in this study top, Gaussian; bottom Gaussian with flare.

Effects of the filters on 10% noise edges is somewhat different from the results of 3% noise edges. Referring to the graphs of comparison data presented below it is evident that the chop filter performs best at all frequencies up to the frequency limit on Gaussian edges without flare. The damping filter doesn't significantly improve MTF results at spatial frequencies below 8 Cpmm.

On Gaussian edges with flare all three edge treatments produce similar MTF data up to 6 Cpmm. Above that frequency level the damping filter generally has the most favorable effect. The chop filter seems to do better around 7 and 8 Cpmm. But this just reflects the transition from being over biased to under biased. Chop filter MTF data at higher frequencies is unreliable when edges have flare.

Examination of the 20 MTF composite plots for 10% edges on the following pages further clarifies what the edge filters are doing. Note that the filters most effectively reduce the variances in MTFs between 10 and 20 Cpmm. The chop filter does this best. But on edges with flair the chop filter again biases the MTF information producing nice looking but very erroneous curves.

The damping filter is the most effective one for edges having flare. However, this is less true at 10% noise than at 3% noise. In fact for triangle spread function edges with flare and 10% noise both filters actually degrade the unfiltered edge MTF below 7 Cpmm. Above that the chop filter performs slightly better than the damping filter. (See triangle data on pages 60 to 75 in the appendix). This loss in effectiveness comes about when the noise amplitude is large relative to the flare amplitude. This suggests that there is a noise limit beyond which these filters lose their effectiveness. The results section will conclude with an analysis of this noise limitation.

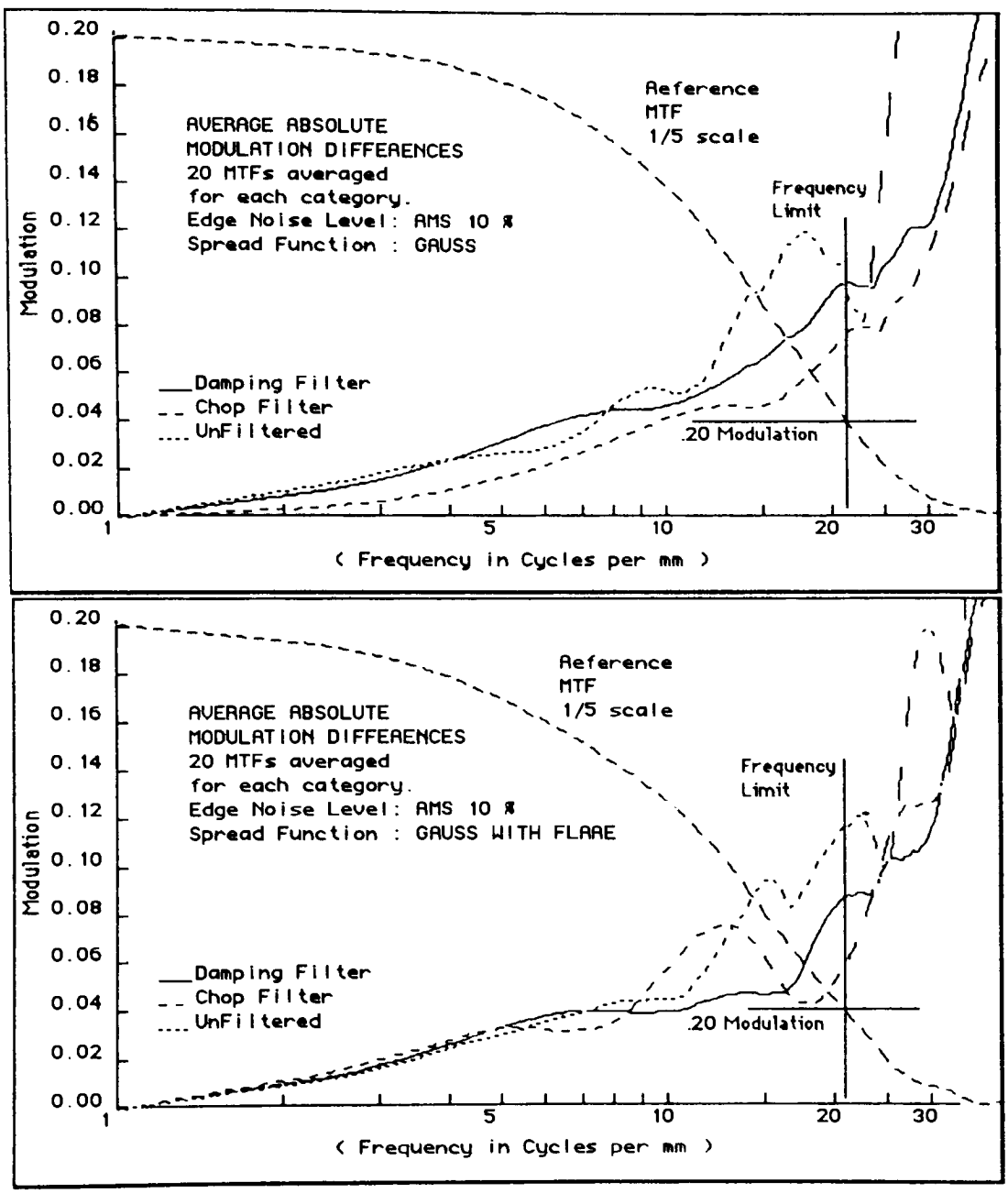


Figure 3.8 Comparison of modulation errors for 3 edge treatments.
Top, Gaussian edges ; bottom, Gaussian edges with flare.

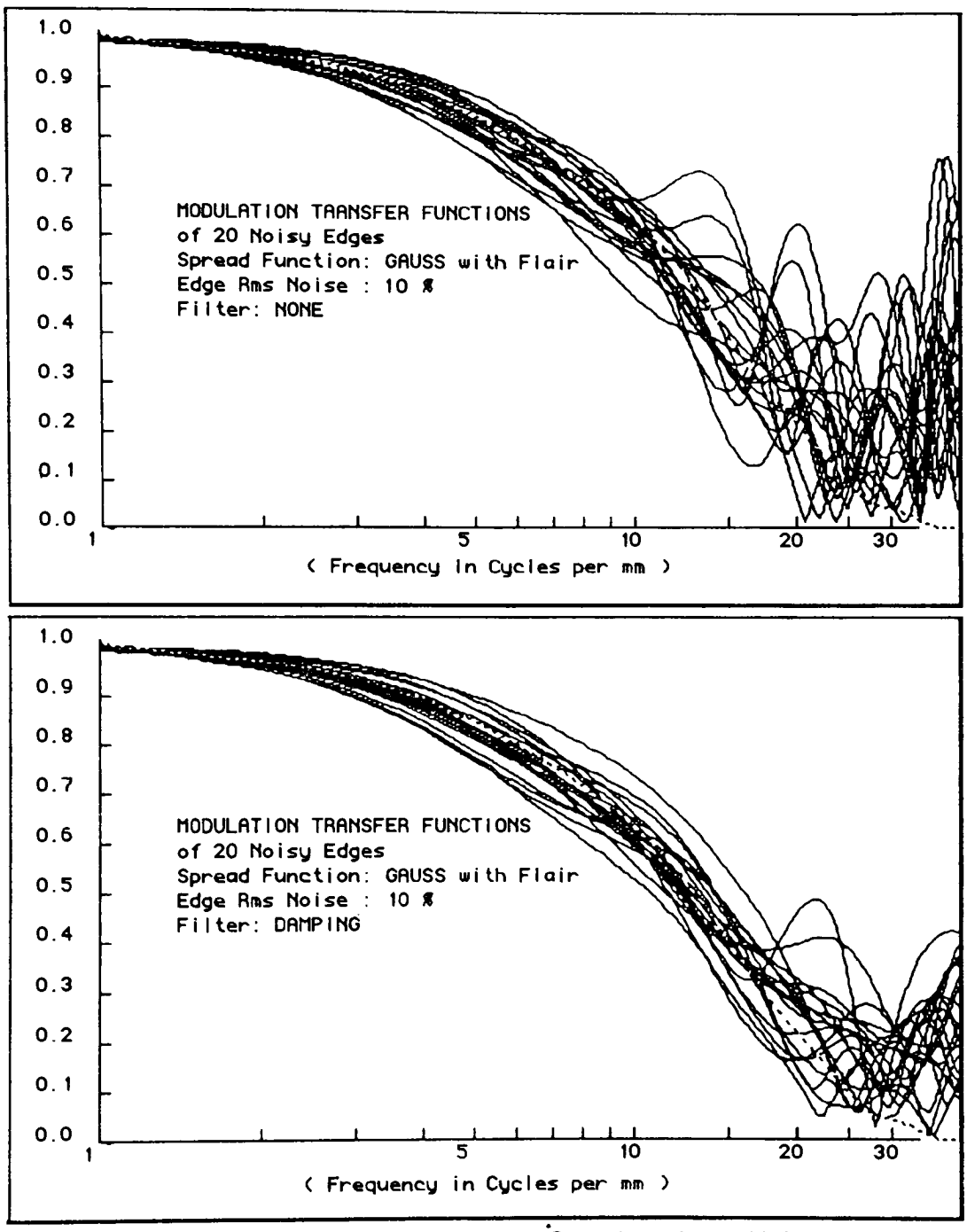


Figure 3.9 Composite plots of 20 MTFs of 10% Gaussian edges with flare. No filter at the top and damping filter at the bottom. Note reference MTF as dashed line.

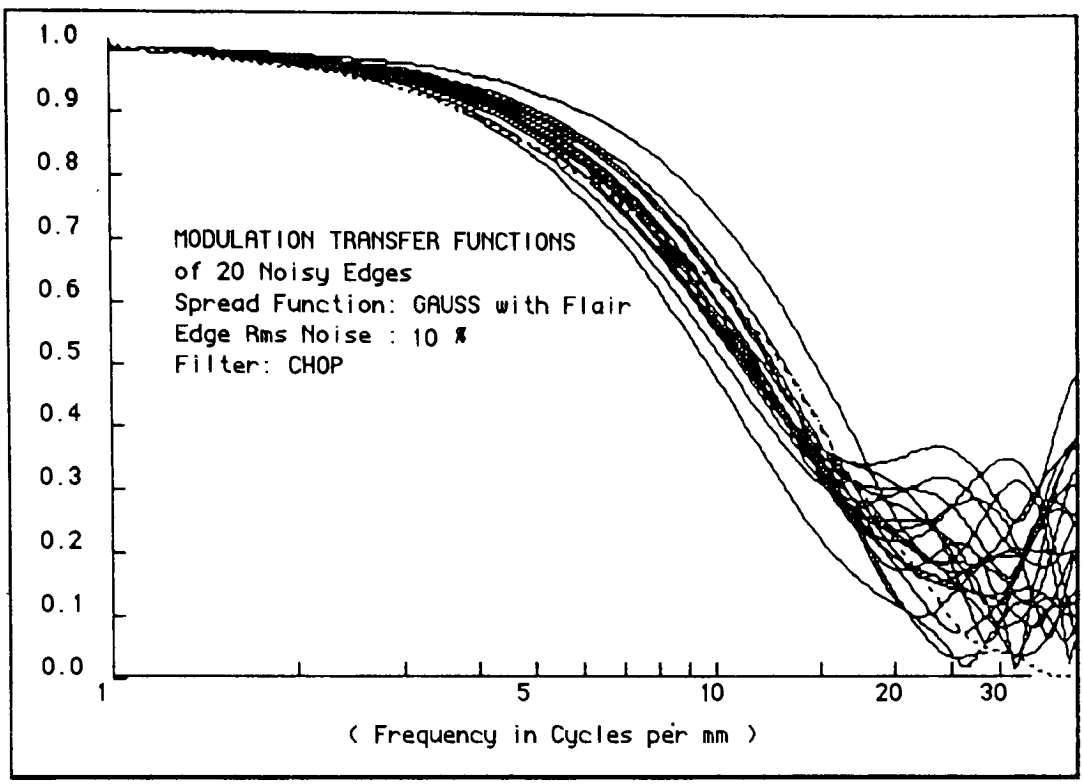


Figure 3.10 Composite plot of 20 MTFs of 10% Gaussian edges with flare. Chop filter used. Note reference MTF as dashed line.

3.1.4 Noise Analysis

Average absolute differences for three noise levels of the same edge functions are plotted on a linear frequency scale in the graphs below. Most of these plots can be approximated by a linear function of appropriate slope beginning near the origin. Extensions of these lines intersect a full scale reference MTF curve. These intersections can be thought of as limiting points where the average modulation error exceeds the modulation. The filters are increasingly less effective as this frequency limit is approached.

Within a reasonable range the relationship between the slope of each line and RMS edge noise is approximately linear. Therefore it is possible to interpolate or extrapolate slopes in an attempt to select a line that would assist in predicting a maximum usable noise level. It is also possible to construct a line to represent MTF errors for a particular noise level. This has been done as an example of a predicted line of MTF errors for a population of 5% noise Gaussian edges in the first three graphs on the next pages. Graphs of Gaussian edges with flare for the 3 edge treatments are also presented for comparison. Similar graphs for triangle edge functions are in the appendix. Because of the distortion discussed previously this technique doesn't work with chop filters on flare edge MTF data. The plots of chop filter MTF errors in the last graph provide evidence for this.

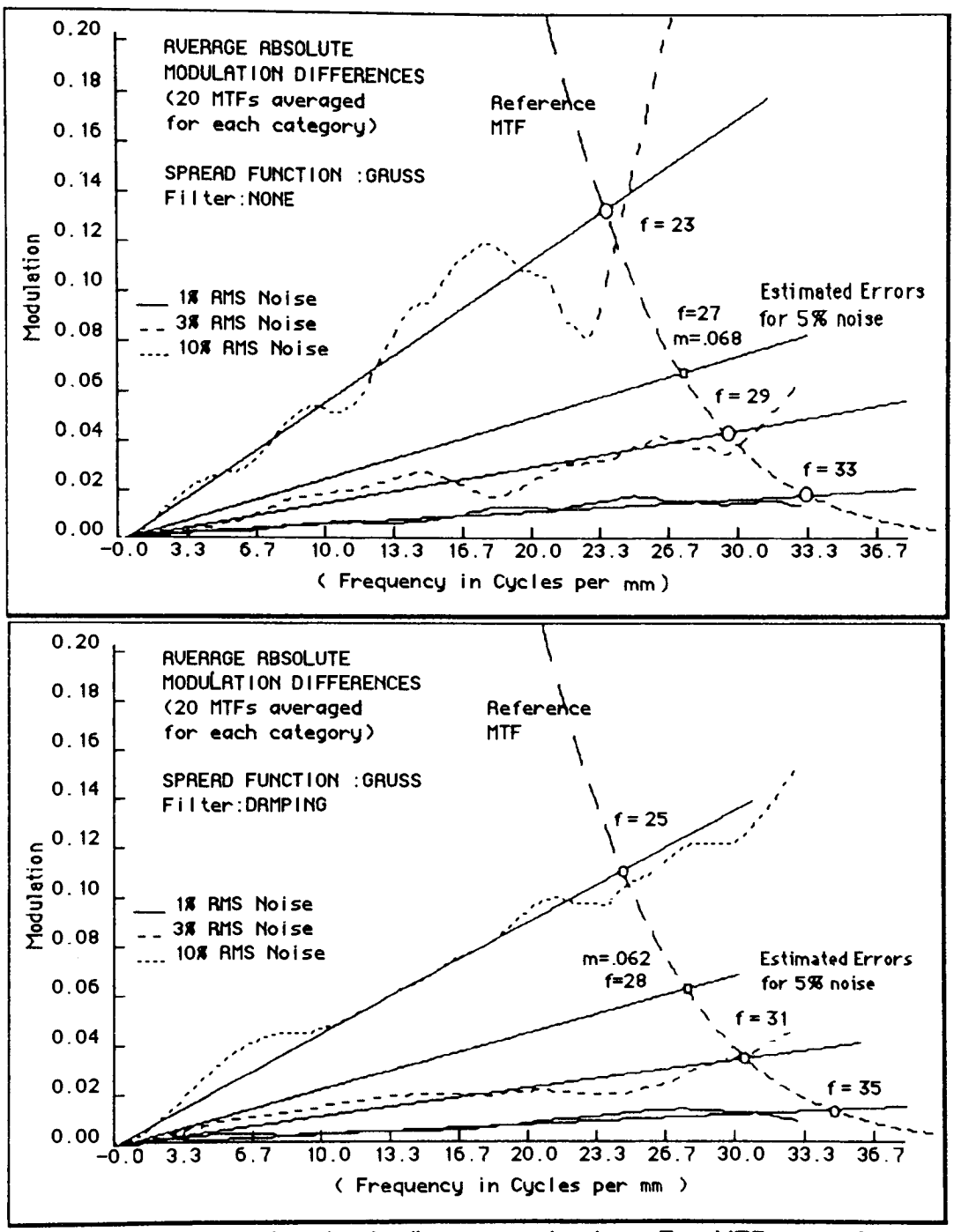


Figure 3.11 MTF error plots showing linear approximations. Top, MTF errors of unfiltered Gaussian edges. Bottom, MTF errors of damping filtered Gaussian edges.

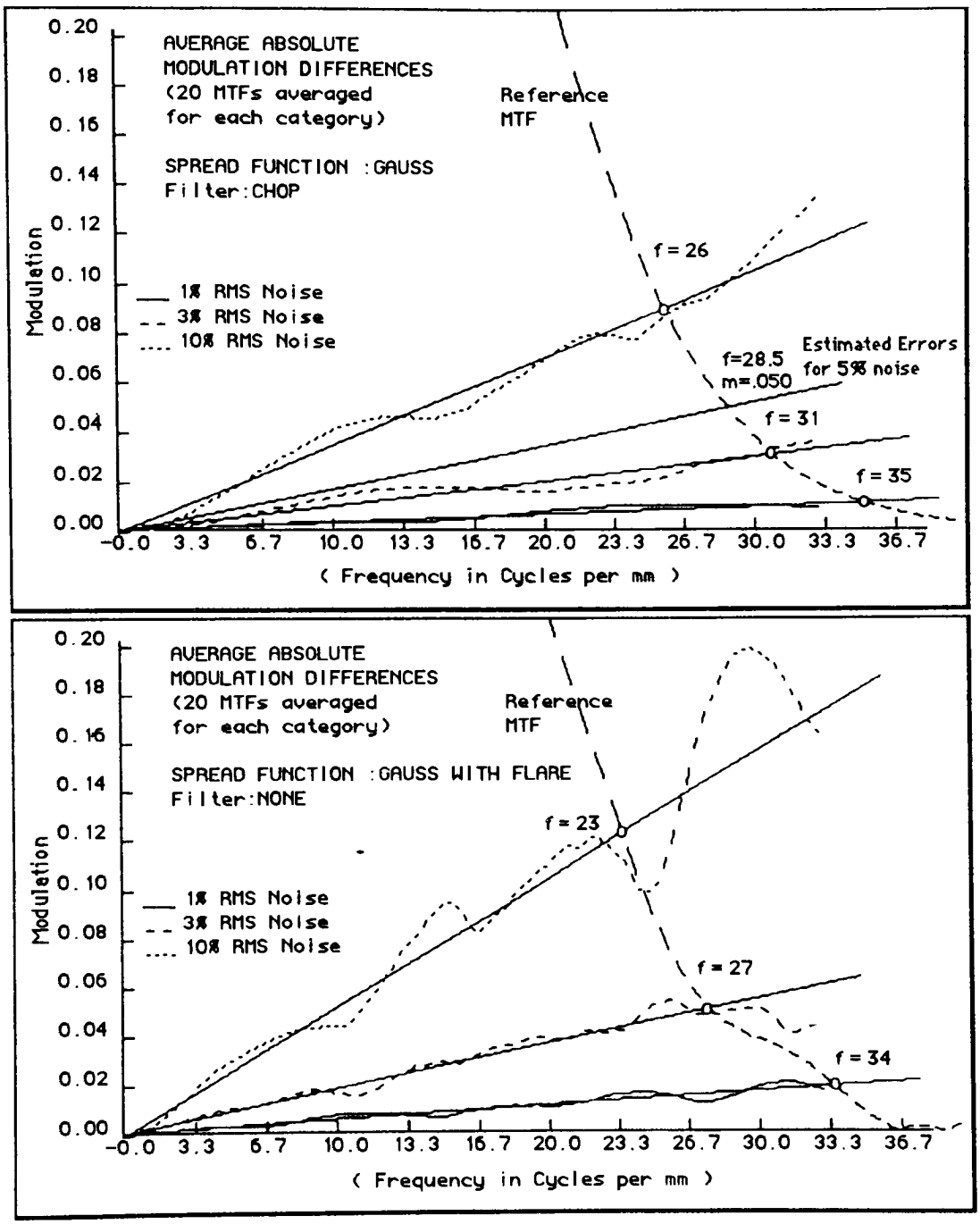


Figure 3.12 MTF error plots showing linear approximations. Top, MTF errors of chop filtered Gaussian edges. Bottom, MTF errors of unfiltered Gaussian edges with flare.

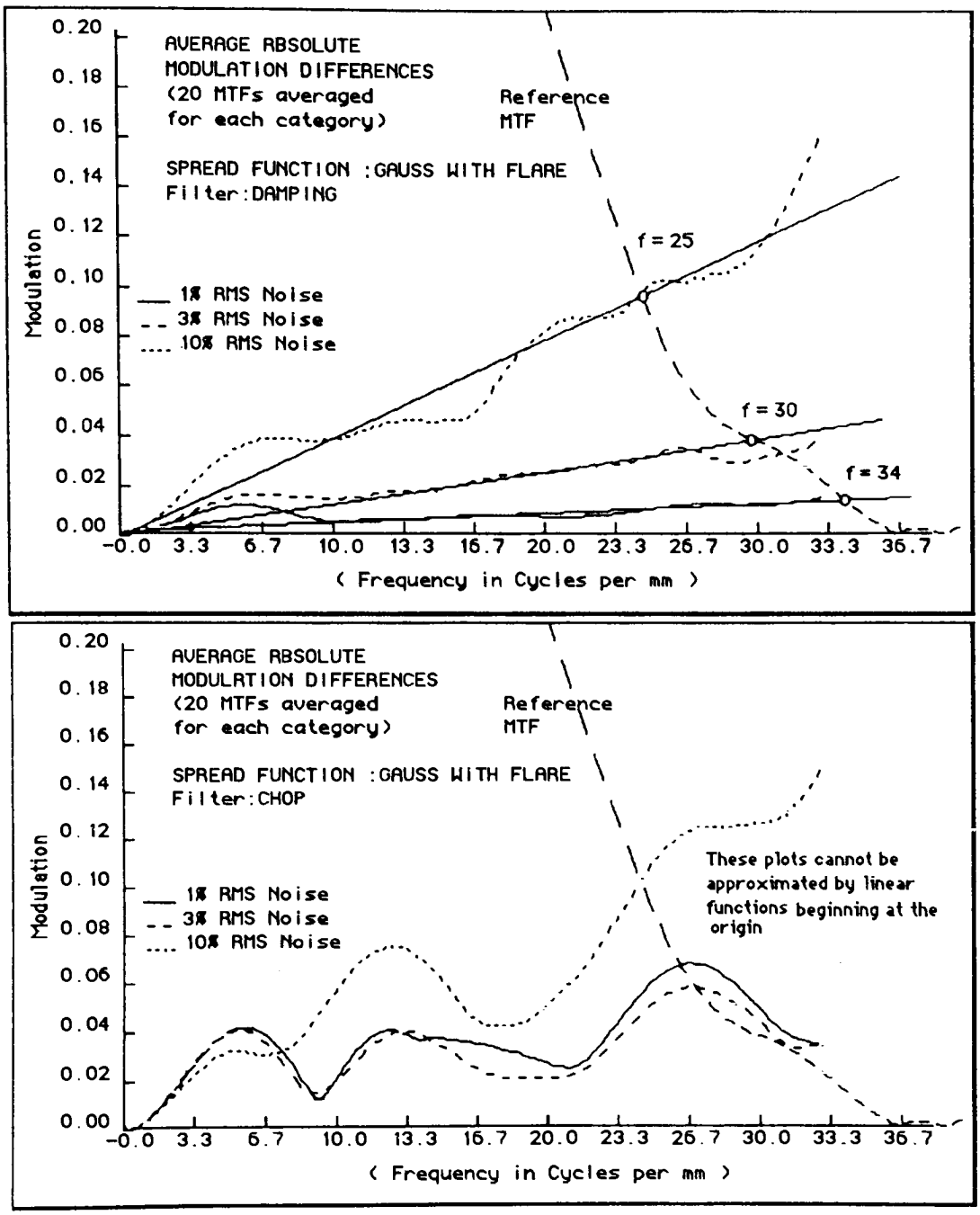


Figure 3.13 MTF error plots. Top, MTF errors of damping filtered Gaussian edges with flare. Bottom, MTF errors of chop filtered Gaussian edges with flare.

4.0 Discussion

It should be emphasized that the conclusions included here apply only to linear imaging systems in those cases where the noise is uncorrelated.

4.1 Six Criteria for a Method of Treating Noisy Edges

On page 7 in the introduction characteristics for an ideal method of finding OTFs from noisy edges are listed. The two edge filters along with Tatians method for transforming edge functions easily meet the first four. A single edge is used, there is no numerical differentiation, discrete data is used, and there is no smoothing of the actual edge information. I will make the case that the remaining two criteria were closely approached if not completely achieved. These are: " Avoid subjective judgments involving parameter selection" and " Return an accurate OTF/MTF".

4.2 Avoid subjective Judgments & Parameter Setting

As stated there was an attempt to design the filters so that they would perform automatically on a variety of edge types without having to reset any numbers. None were reset for any of the four types of edges considered here. However the arc sine method for establishing toe and shoulder points does not match the triangle edge quite as well as the Gaussian edge. Perhaps a different offset could have been used for establishing the toe and shoulder points for triangle edges. However, any errors incurred did not seem to have a significant effect on MTF output. This might not be so for edges having a rectangular spread function for instance. This then may be one criterion that was not completely achieved.

4.3 Return an Accurate OTF/MTF

The question here is: "does the shift variant filter increase the accuracy in MTF that can normally be achieved from a noisy edge?" The answer is yes for most but not all cases. Both filters have improved MTF accuracy in two ways. First the MTF error introduced by noise is usually reduced, second the variation of errors (99% confidence interval) is always reduced by filtering. The exception to these foregoing statements occurs when edges have flare and low noise levels (near 1% RMS). Unfiltered edges resulted in the lowest error means. For all other cases studied in this thesis one or both of the two filters provided some improvement in mean error; as much as 46% by the chop filter on 10% noise Gaussian edges.

The chop filter proved to be superior in dealing with the noise in edges with simple spread functions. However the best overall filter is the shift variant (damping) filter since it improves MTF accuracy for the most classes of edges, particularly edges with flare where the chop filter is unreliable. This suggests that when there is little foreknowledge about the nature of a relatively noisy edge the damping filter can be relied on to always provide improvement in MTF data but not the chop filter.

4.4 Suggestions for Further Work

An interesting study would be to use the chop and damping filter on real edge data and compare the results with previously used methods such as the averaging method used by Cadou. However there would be no way of knowing what the actual OTF for the data would be and therefore no standard for comparison.

Another possible way to use real data and also to achieve a comparison standard would be to construct a real edge and photograph it or use it as the objective in an enlarger. Different spread functions could be introduced by defocusing known amounts. Noise could be added by using a range of fine grain to course grain photographic materials.

Another project would involve testing the feasibility of using the filter as part of a CCD linear array camera system that would display the MTF resulting from scanning an edge.

Finally the suggestion that MTF error characteristics for different noise levels can be predicted using a linear approximation of a known MTF error function needs to be tested.

Appendix I

Simulation Parameters

Edges

Number of data points	256
Sampling increment	.0012mm
Nominal width of spread Function	60 points (.072mm)
Noise levels (RMS%)	1 , 3 , 10
Number of samples at each noise level	20

Filter

Smoothing averaging block length	60 points
Shift Vr Filter Min. Avng. Blk. length	1 point (2Q-1)
Shift Vr Filter Max. Avng. Blk. length	90 points (2Q-1)

Modulation Transfer Functions

Number of data points	120
Frequency increment	.33 c/mm
Nyquist Frequency	416.7 c/mm

APPENDIX II

GAUSSIAN SPREAD FUNCTION TYPE EDGES

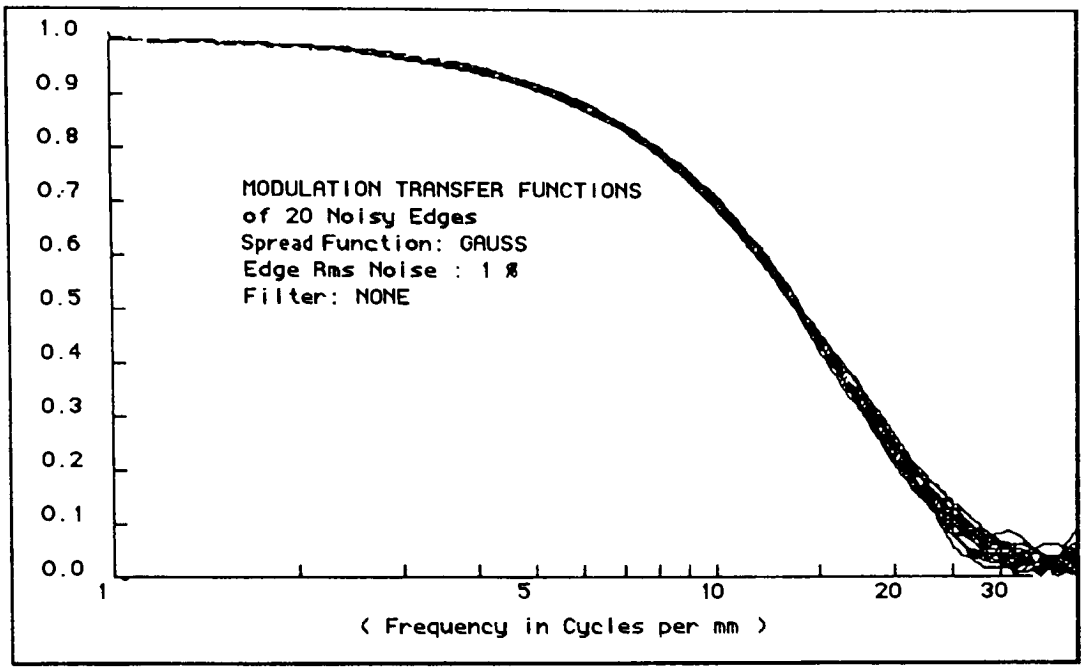
RESULTS DATA

(Includes any Gaussian data
not found in the results
section on pages 29 to 34.)

Including:

Modulation Transfer Function Composite Graphs

Average Absolute MTF Errors Graphs



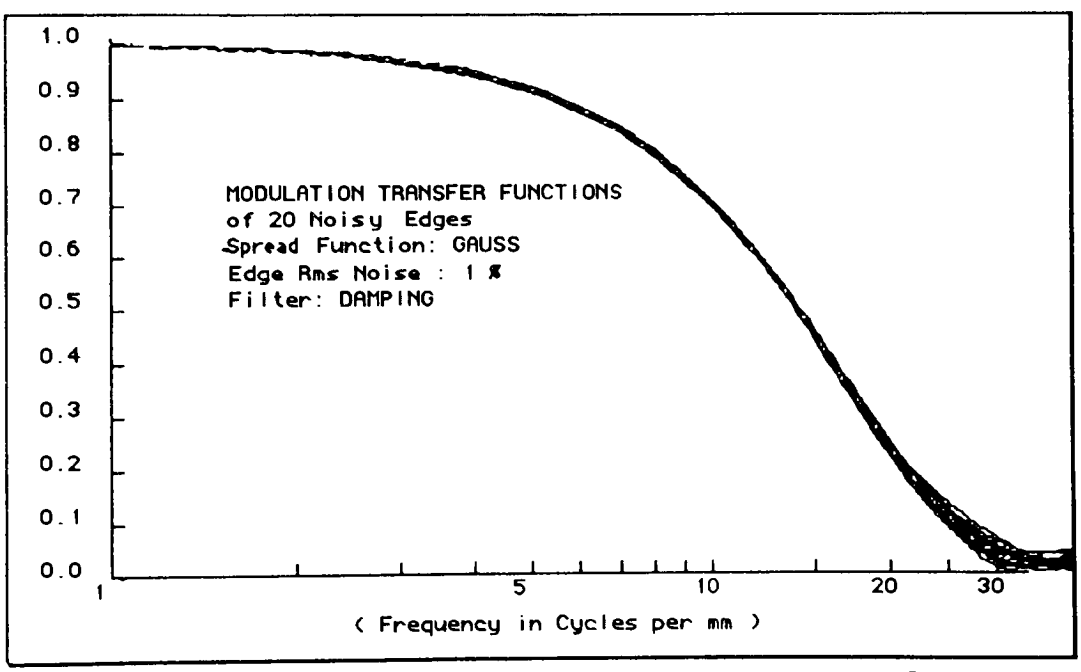


Figure AII.1 Composite plots of 20 Modulation Transfer Functions of 1% RMS noise edges generated from a Gaussian spread function; top no filter, bottom damping filter. Dotted line traces the noiseless reference MTF.

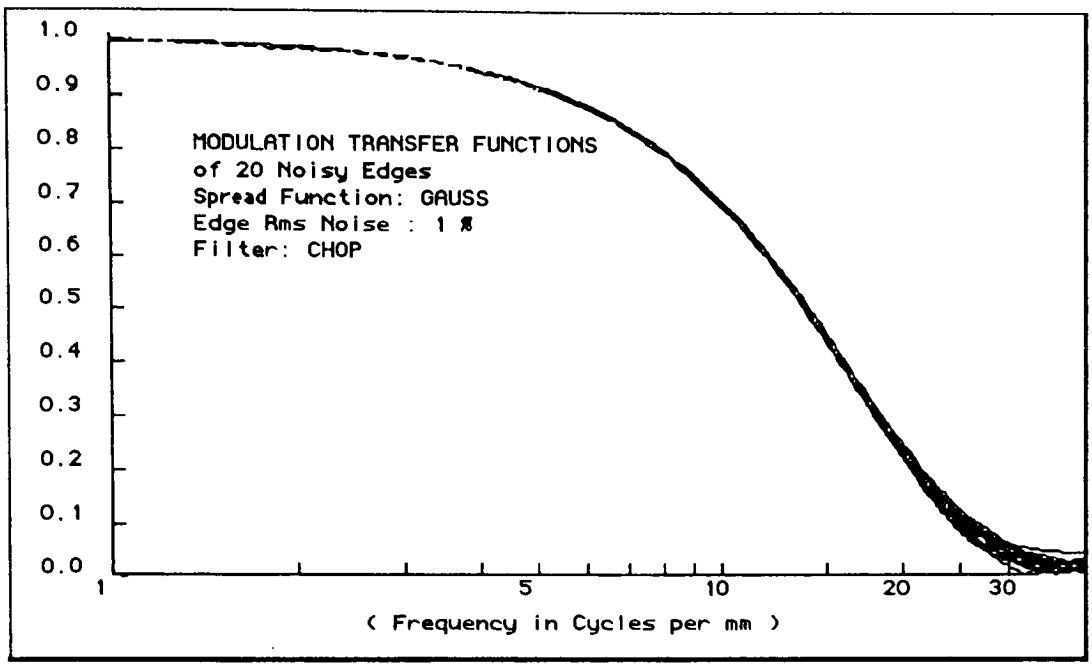


Figure AII.2 Modulation Transfer Functions for 20 1% RMS noise edges generated from a Gaussian spread function. Edges were chop filtered. Dotted line traces the noiseless reference MTF.

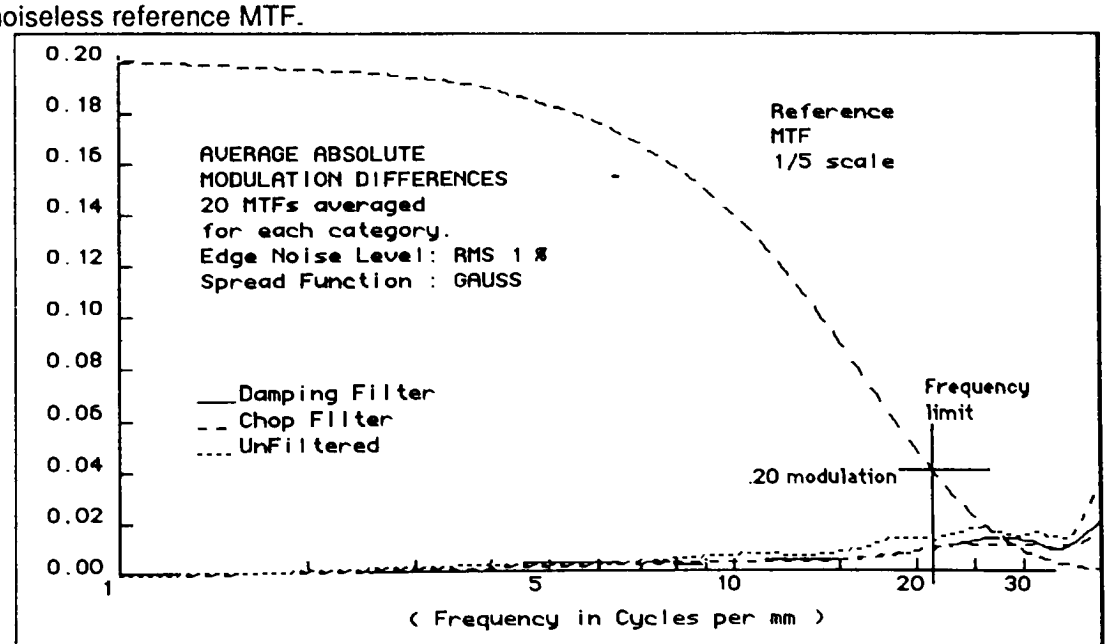


Figure AII.3 Plot of MTF errors compared to a noiseless reference for 3 edge treatments. Gaussian type edges at 1% RMS noise.

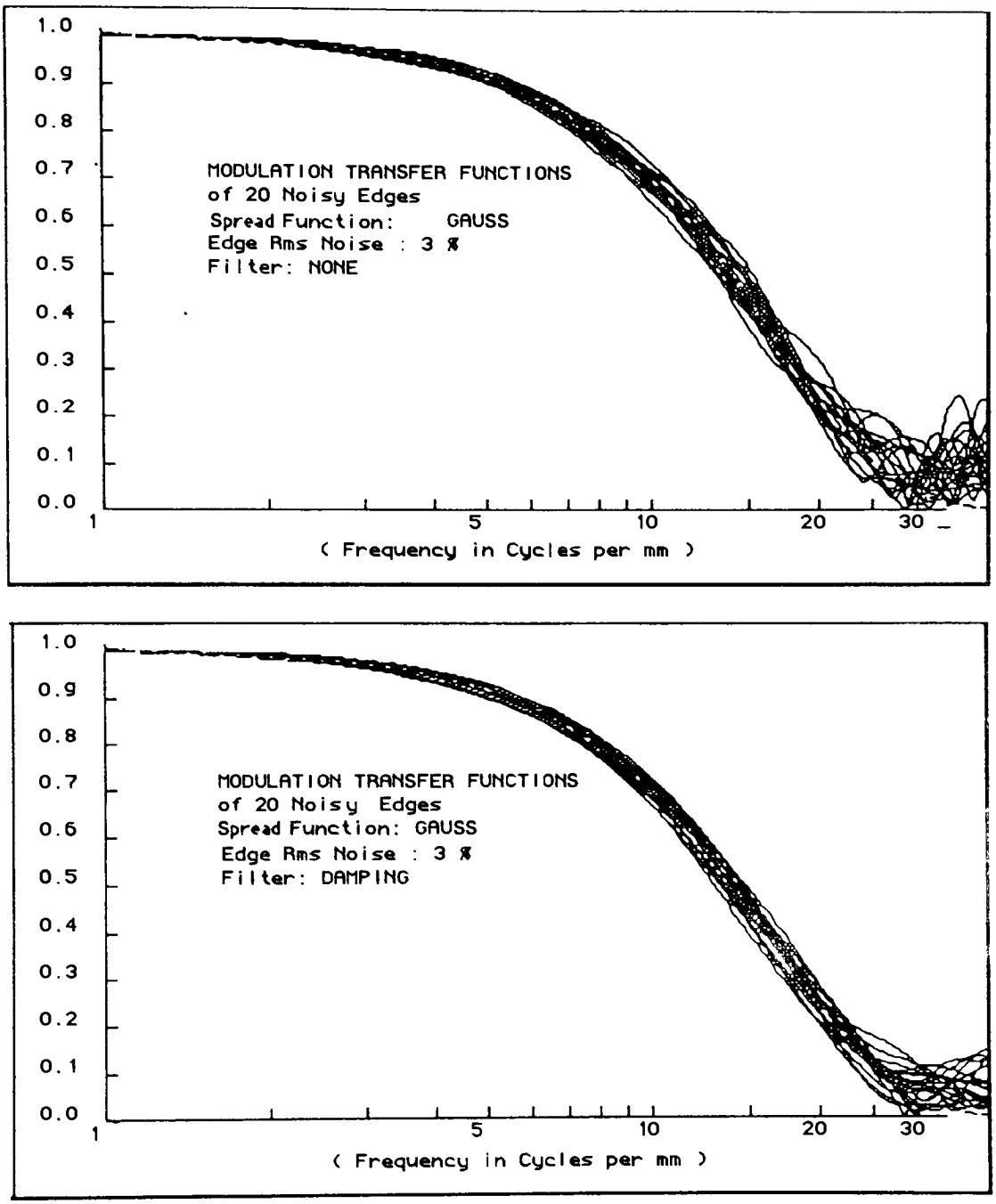


Figure AII.4 Composite plots of 20 Modulation Transfer Functions of 3% RMS noise edges generated from a Gaussian spread function; top no filter, bottom damping filter. Chop filter Composite Graphs and Comparison Graphs for the 3% noise Gaussian edges are omitted here and can be found on pages 31 to 40 respectively.

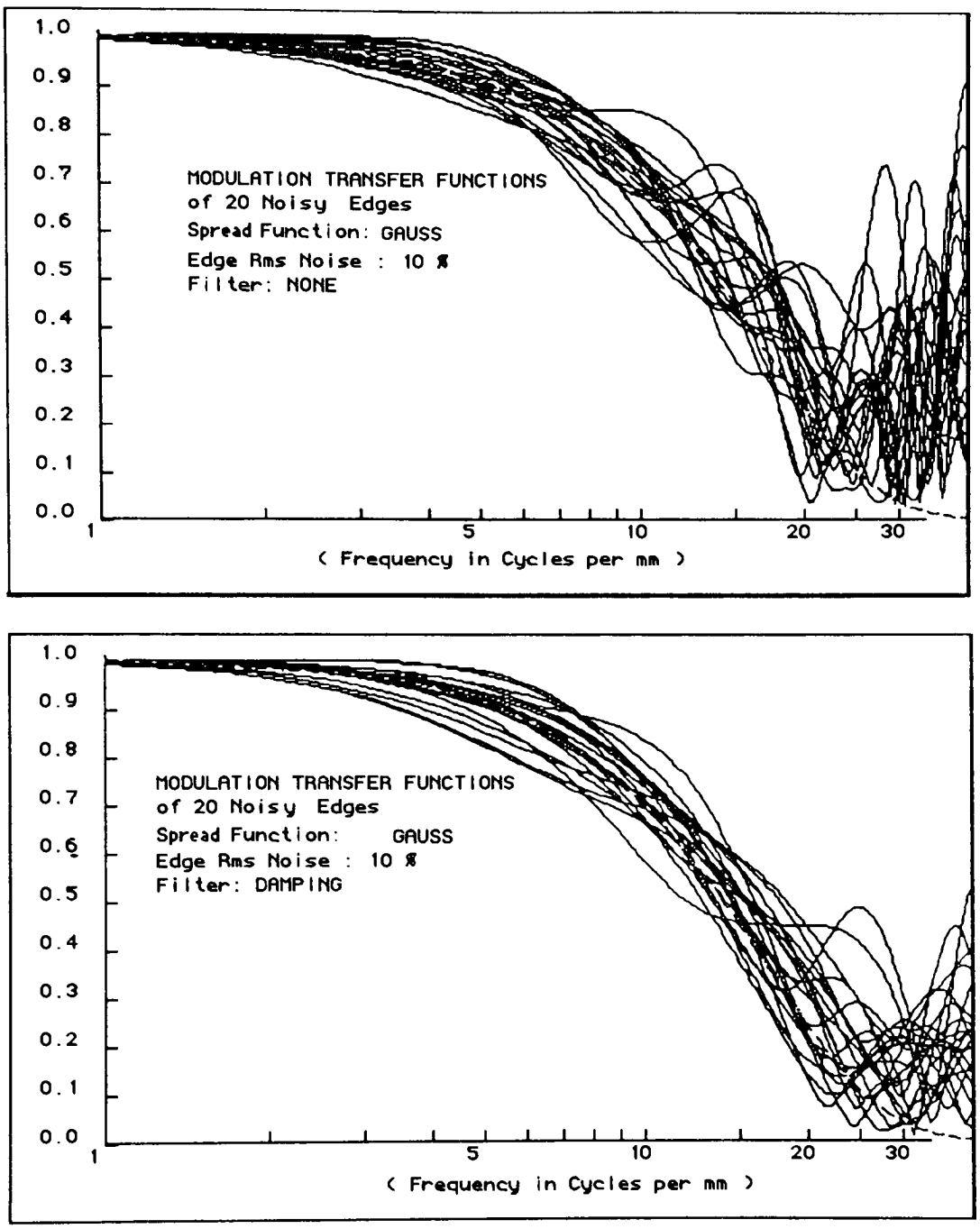


Figure AII.5 Composite plots of 20 Modulation Transfer Functions of 10% RMS noise edges generated from a Gaussian spread function; top no filter, bottom damping filter. Dotted line traces the noiseless reference MTF.

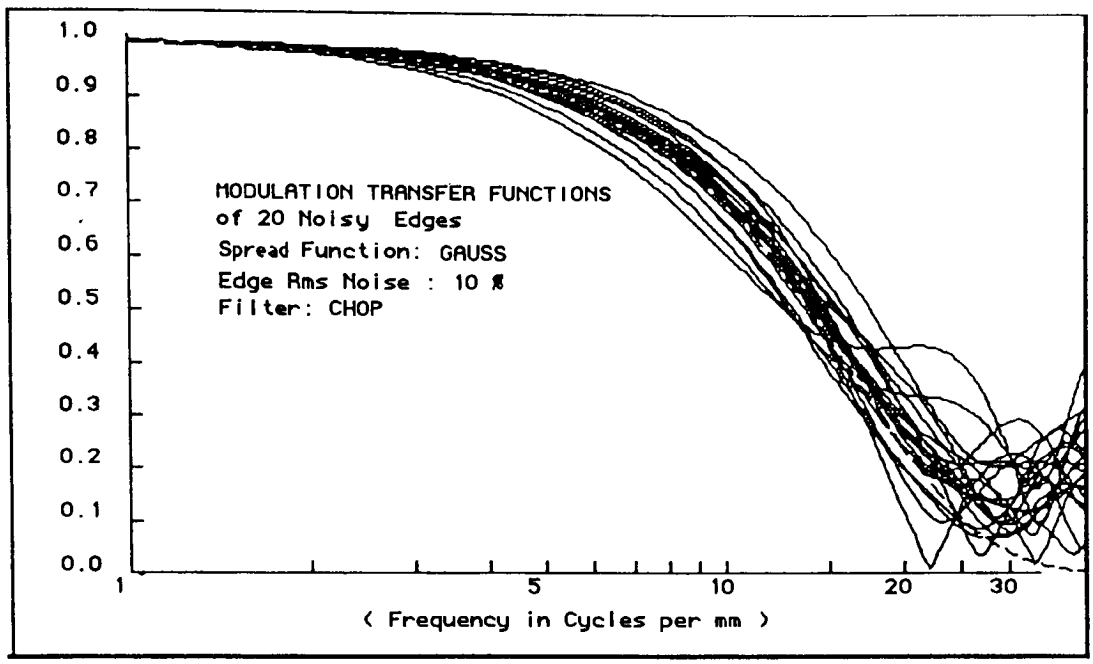


Figure AII.6 Composite plots of 20 Modulation Transfer Functions of 10% RMS noise edges generated from a Gaussian spread function. Chop filter was used. Dotted line traces the noiseless reference MTF.

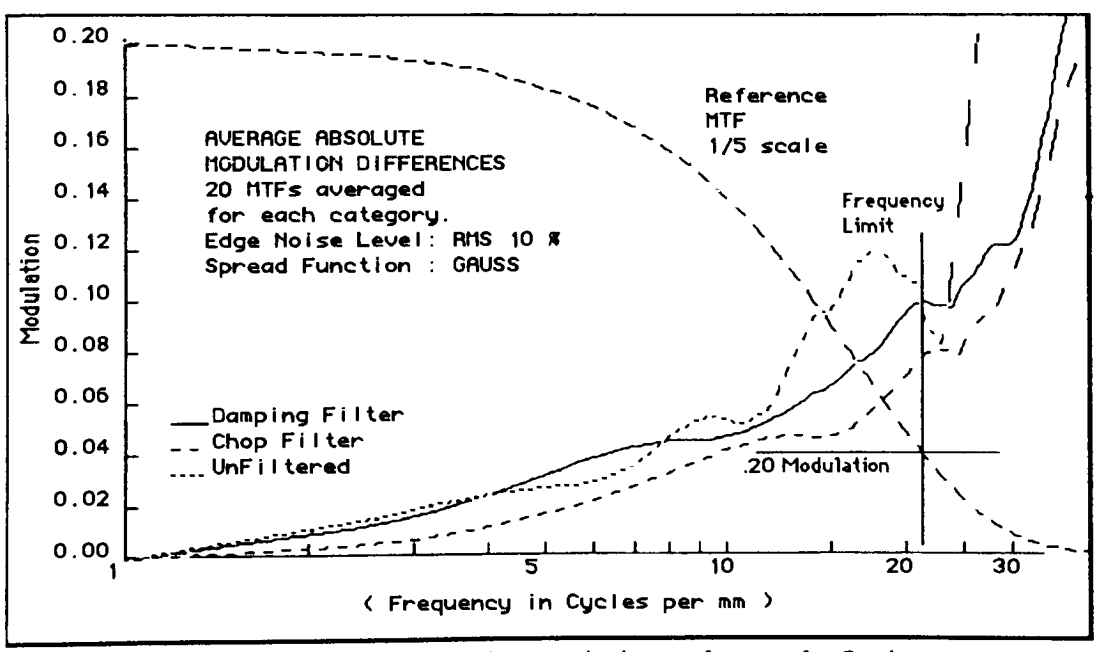


Figure AII.7 Plot of MTF errors compared to a noiseless reference for 3 edge treatments. Gaussian type edges at 10% RMS noise.

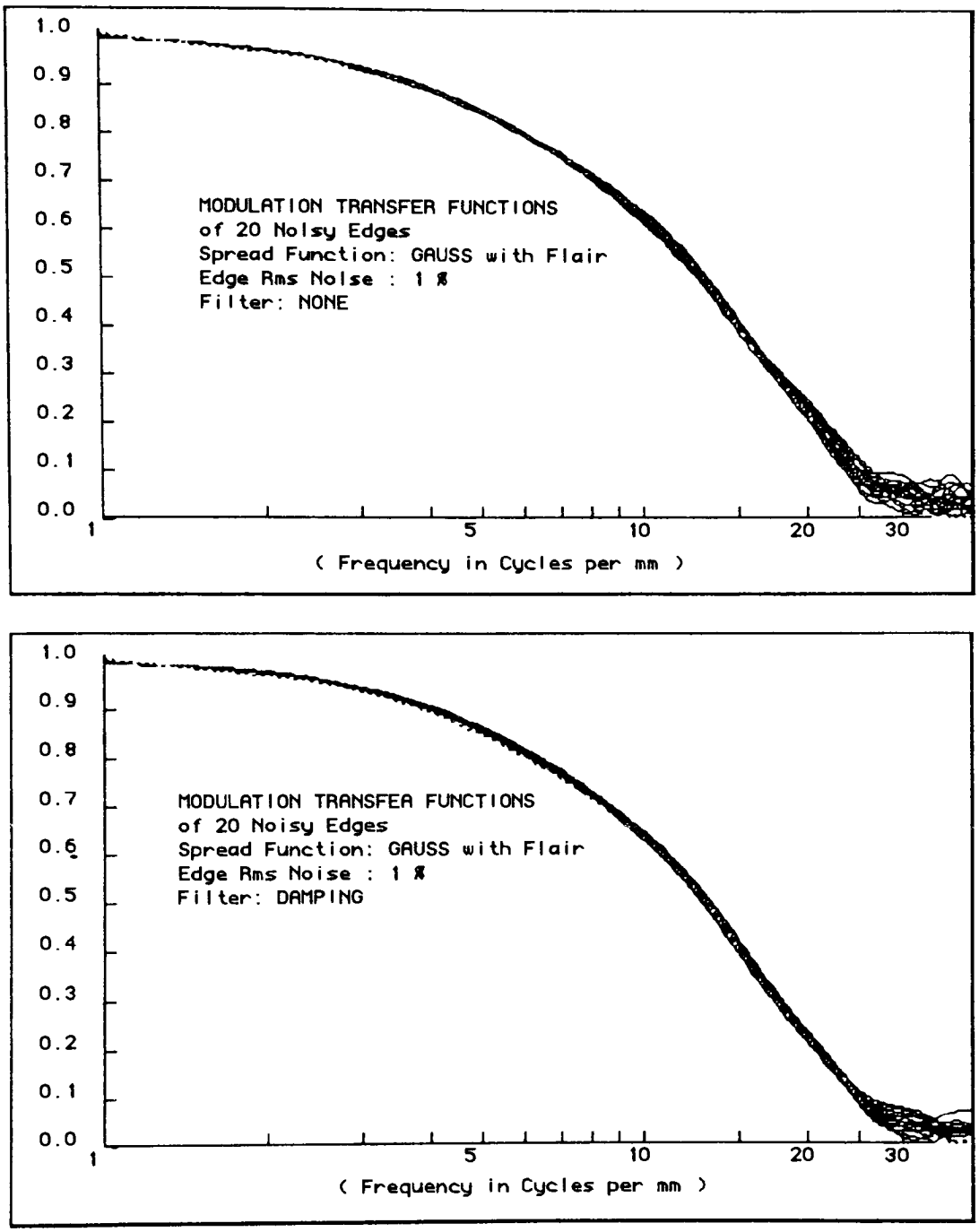


Figure AII.8 Composite plots of 20 Modulation Transfer Functions of 1% RMS noise edges generated from a Gaussian spread function with flare; top no filter, bottom damping filter. Dotted line traces the noiseless reference MTF.

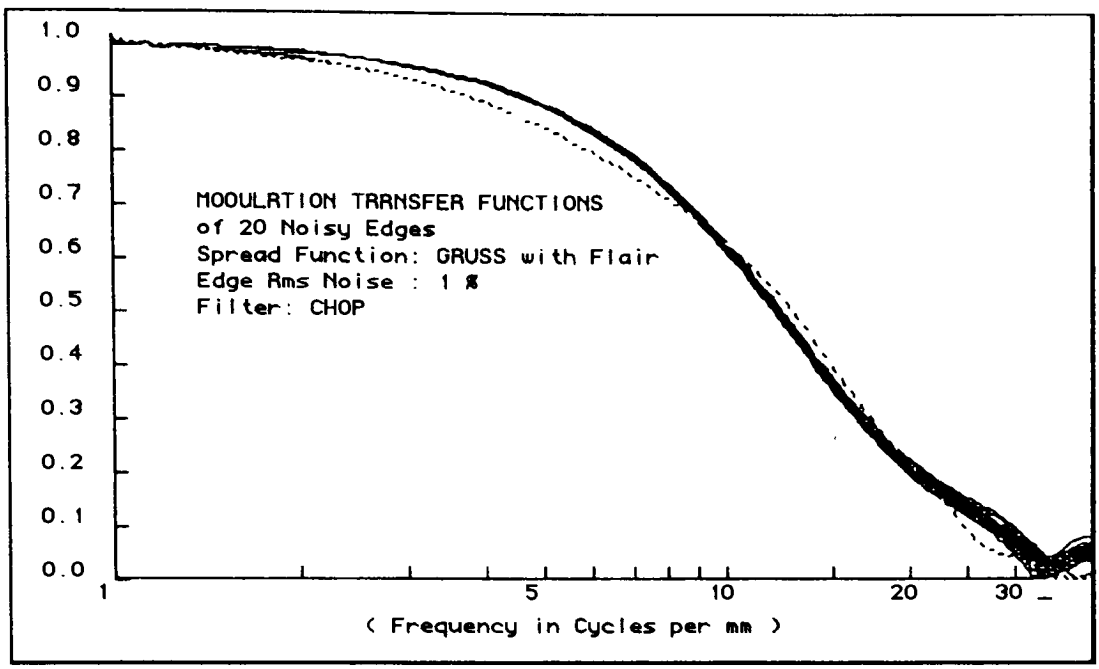


Figure AII.9 Composite plots of 20 Modulation Transfer Functions of 1% RMS noise edges generated from a Gaussian spread function with flare. Chop filter was used. Dotted line traces the noiseless reference MTF.

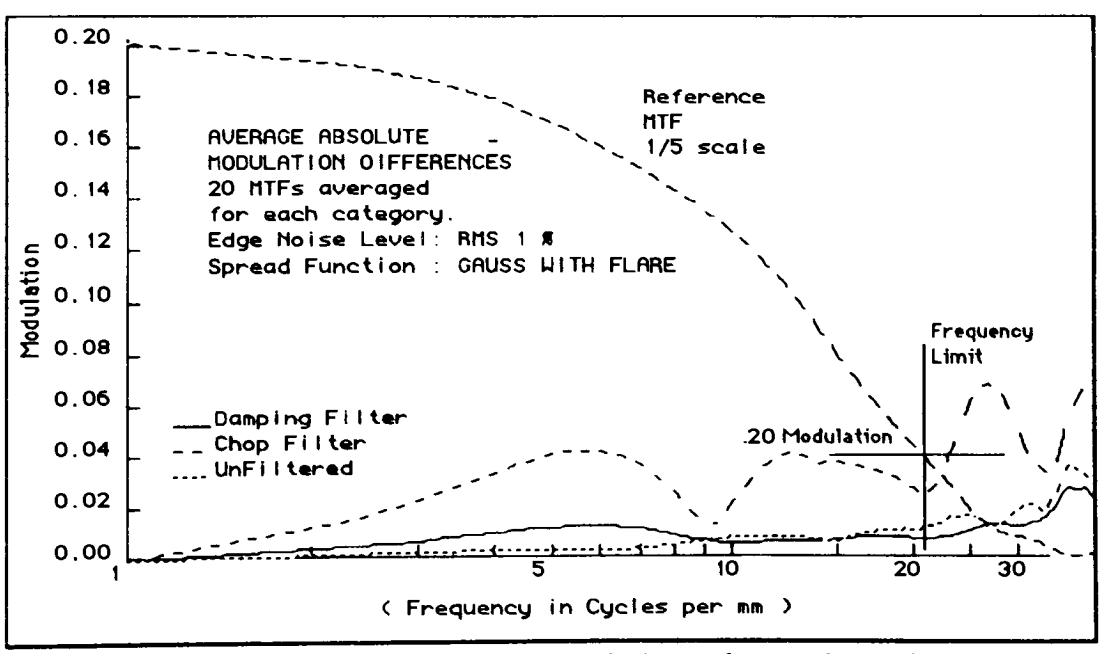
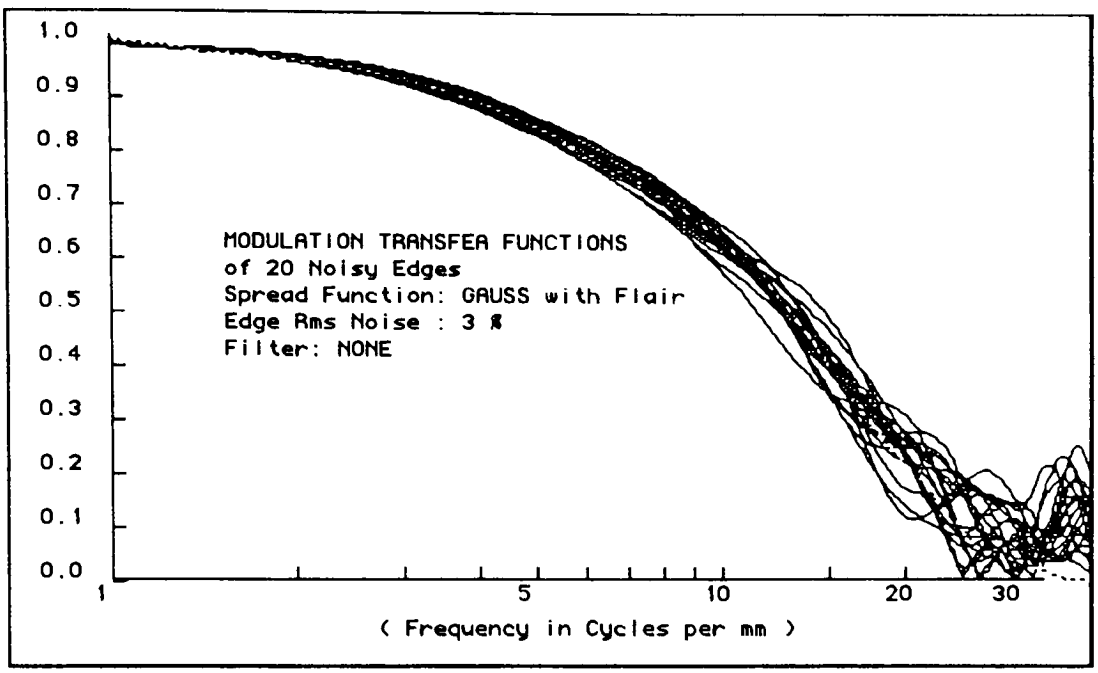


Figure AII.10 Plot of MTF errors compared to a noiseless reference for 3 edge treatments. Gaussian with flare type edges at 1% RMS noise.



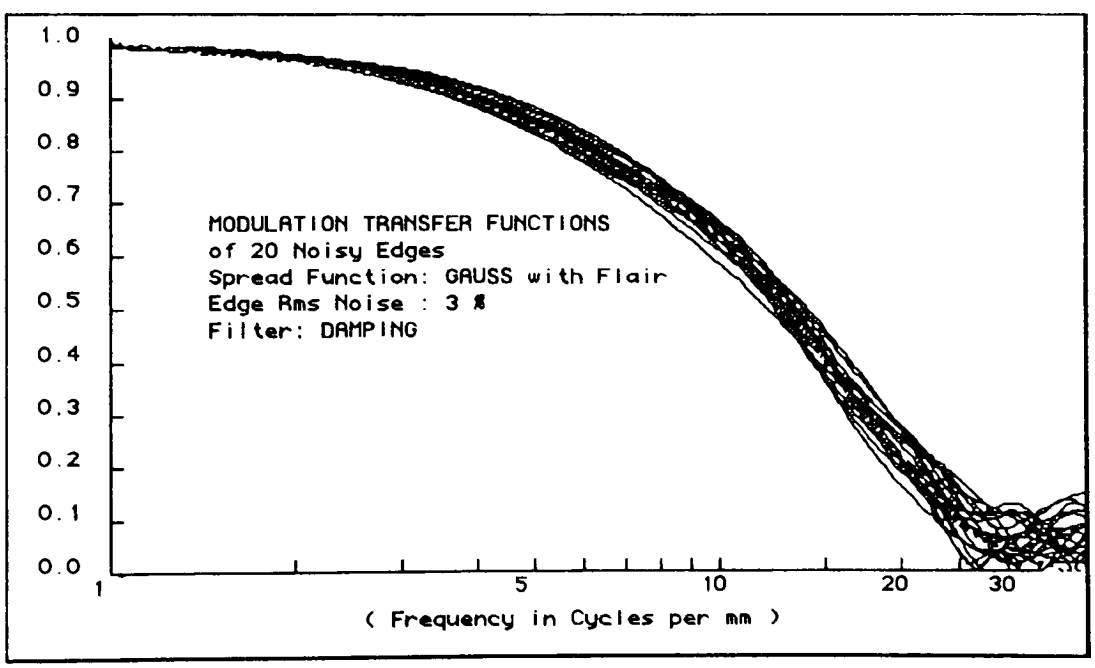


Figure AII.11 Composite plots of 20 Modulation Transfer Functions of 3% RMS noise edges generated from a Gaussian spread function with flare; top no filter, bottom damping filter. Chop filter Composite Graphs and Comparison Graphs for the 3% noise Gaussian edges with flare are omitted here and can be found on pages 31 to 40 respectively.

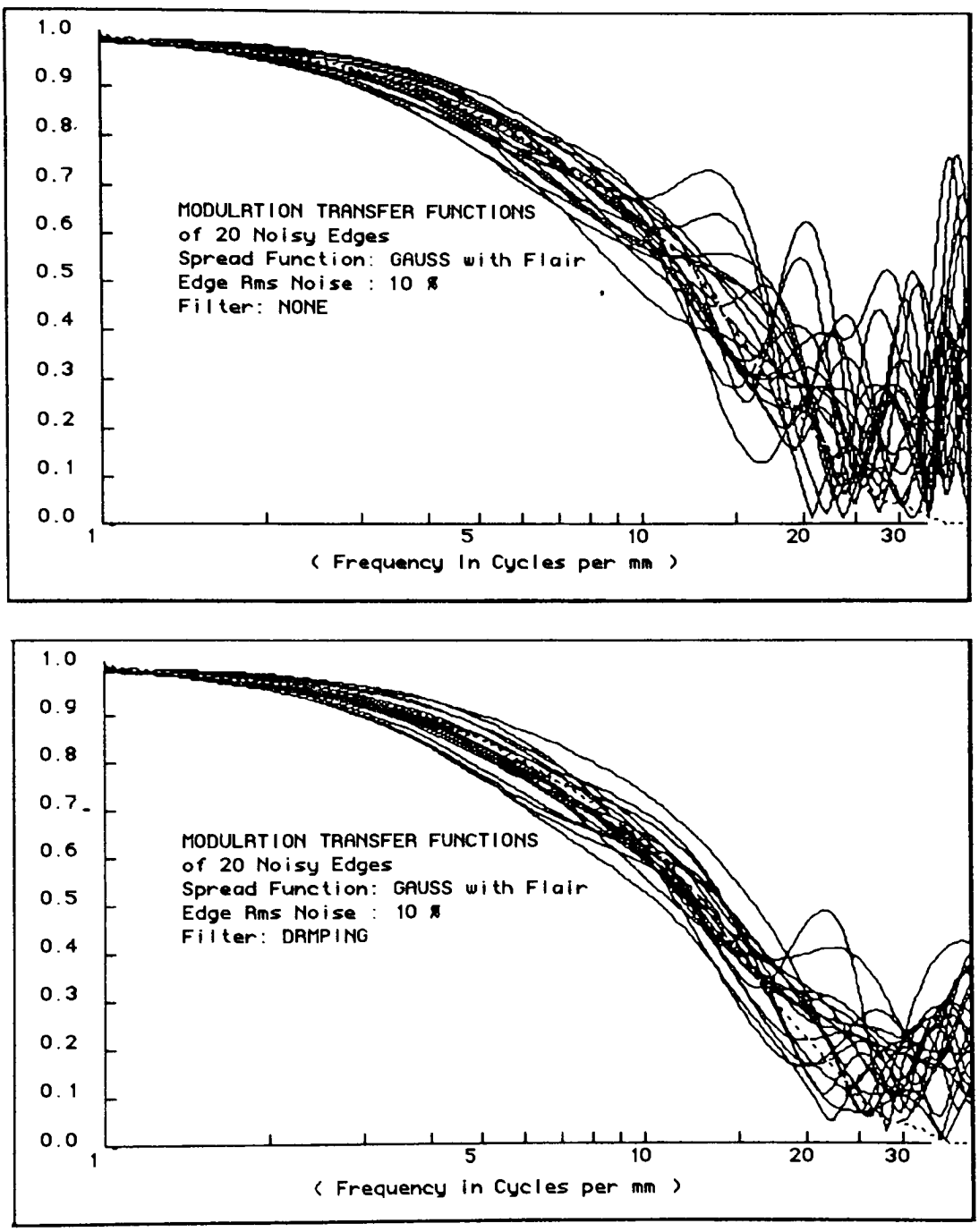


Figure AII.12 Composite plots of 20 Modulation Transfer Functions of 10% RMS noise edges generated from a Gaussian spread function with flare; top no filter, bottom damping filter. Dotted line traces the noiseless reference MTF.

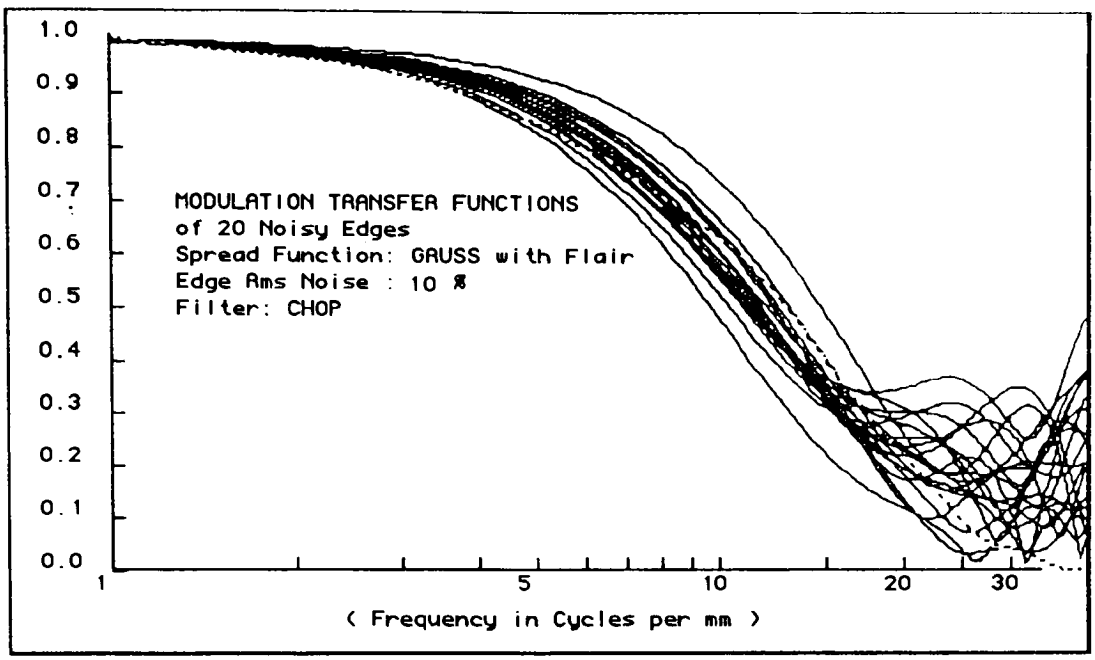


Figure AII.13 Composite plots of 20 Modulation Transfer Functions of 10% RMS noise edges generated from a Gaussian spread function with flare. Chop filter was used. Dotted line traces the noiseless reference MTF.

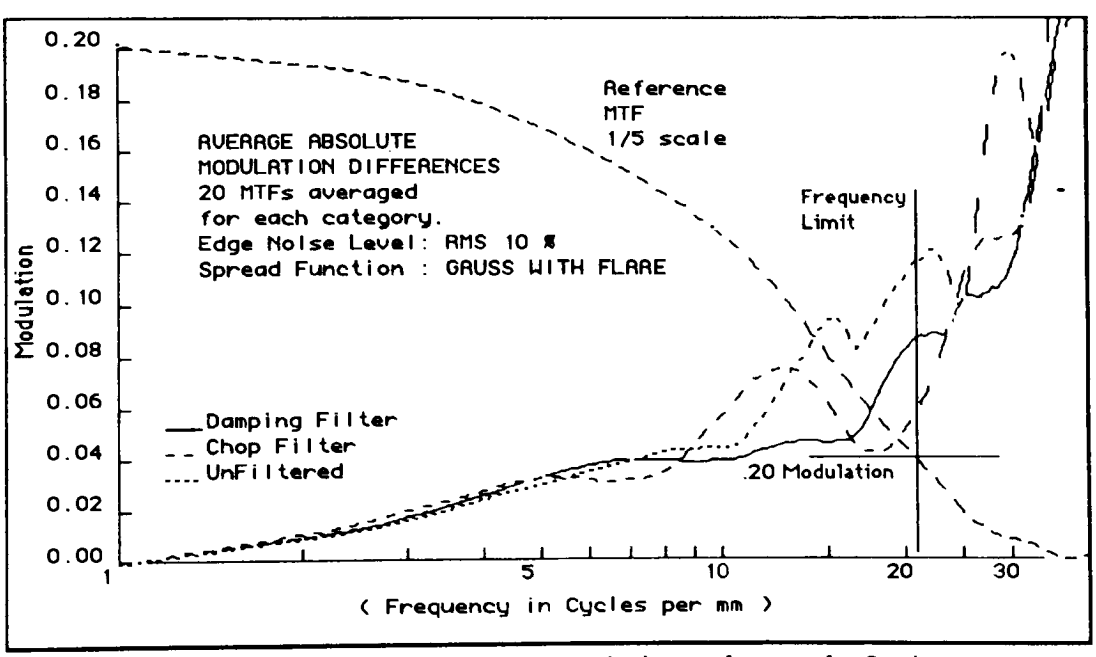


Figure AII.14 Plot of MTF errors compared to a noiseless reference for 3 edge treatments. Gaussian with flare type edges at 10% RMS noise.

APPENDIX III

TRIANGLE SPREAD FUNCTION TYPE EDGES

RESULTS DATA

Including:

Modulation Root Mean Square Differences Table

99% Confidence Interval Graphs

Modulation Transfer Function Graphs

Average Absolute MTF Errors Graphs

ROOT MEAN SQUARE MODULATION DIFFERENCES
COMPARING MTFS OF NOISY EDGES WITH NOISE FREE EDGE MTFS
TRIANGLE & TRIANGLE WITH FLARE

Edge Type Spread Function & RMS Noise	Filter	From 20 to 100% Modulation Mean %	Standard Dev. %	99% confidence Interval Limits
Triangle				
1%	None	0.090	0.037	.066 to .114
1%	Damping	0.069	0.037	.045 to .093
1%	Chop	0.055	0.031	.035 to .075
3%	None	0.253	0.112	.179 to .327
3%	Damping	0.193	0.095	.131 to .255
3%	Chop	0.124	0.077	.073 to .175
10%	None	0.810	0.251	.700 to 1.322
10%	Damping	0.673	0.223	.382 to .984
10%	Chop	0.628	0.253	.347 to .833
Triangle + Flare				
1%	None	0.087	0.041	.060 to .114
1%	Damping	0.101	0.039	.075 to .127
1%	Chop	0.374	0.027	.365 to .392
3%	None	0.274	0.141	.181 to .376
3%	Damping	0.240	0.128	.156 to .324
3%	Chop	0.395	0.065	.352 to .438
10%	None	0.810	0.251	.645 to .975
10%	Damping	0.673	0.223	.527 to .819
10%	Chop	0.628	0.253	.462 to .794

Figure AIII.1 Table of root mean square differences (RMSD) of Modulation Transfer Functions of Triangle type edges. 20 samples per category.

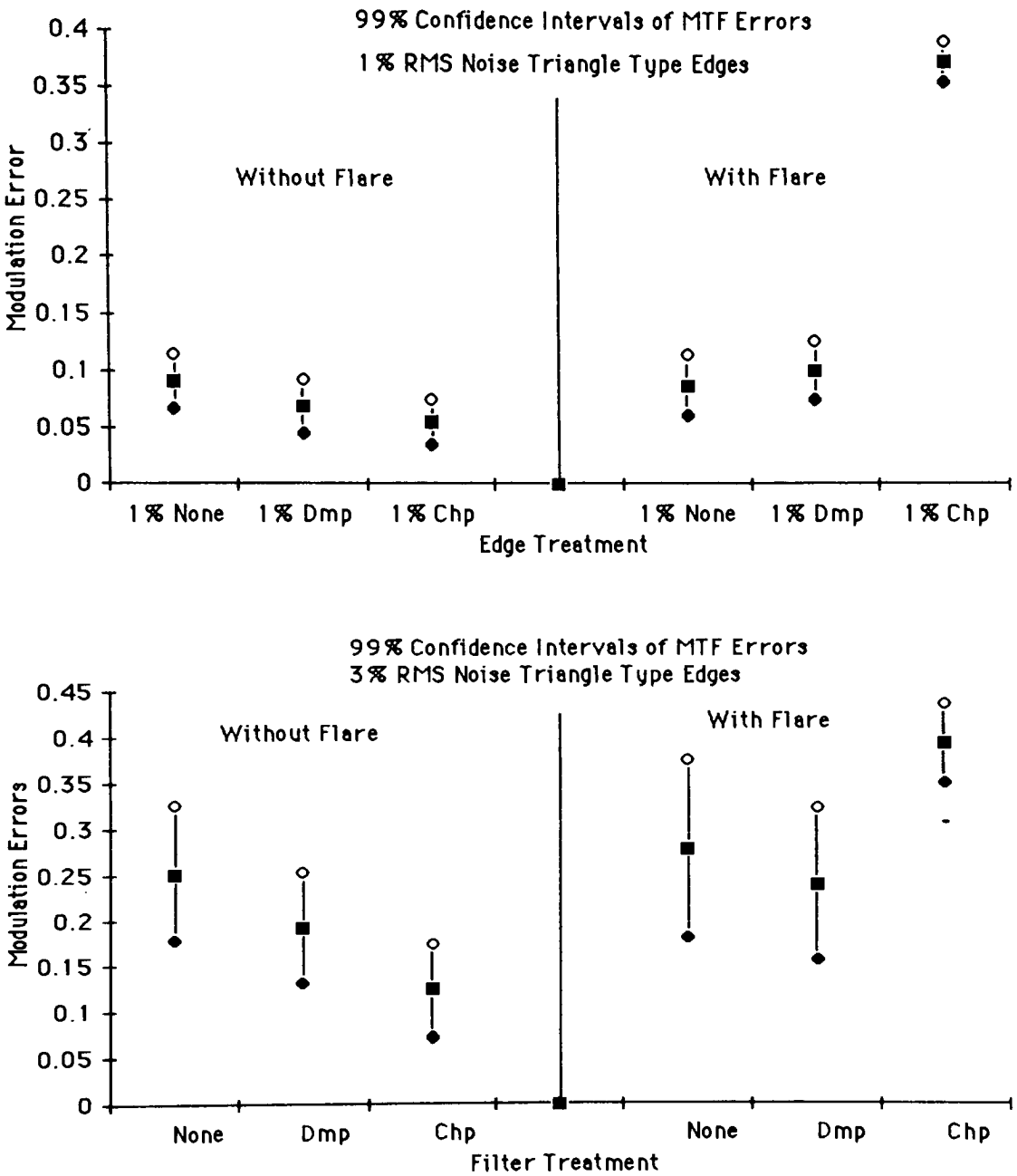


Figure AIII.2 99% confidence intervals for MTF RMS Differences. Filter treatments: none, damping (Dmp.), and chop (Chp.). Top, 1% RMS Noise level. Bottom, 3% RMS Noise level.

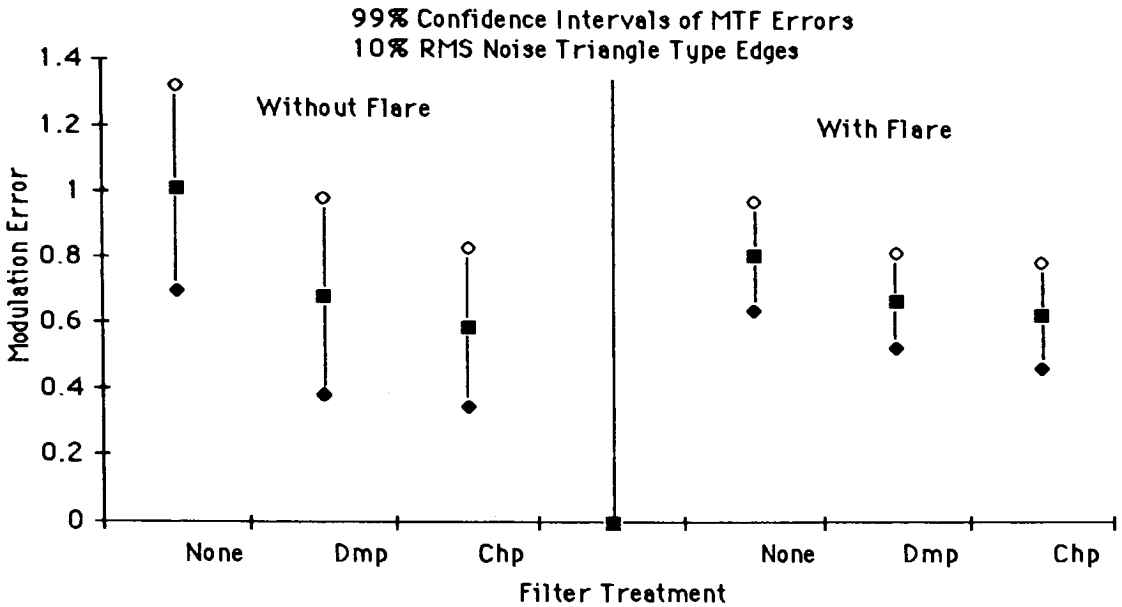
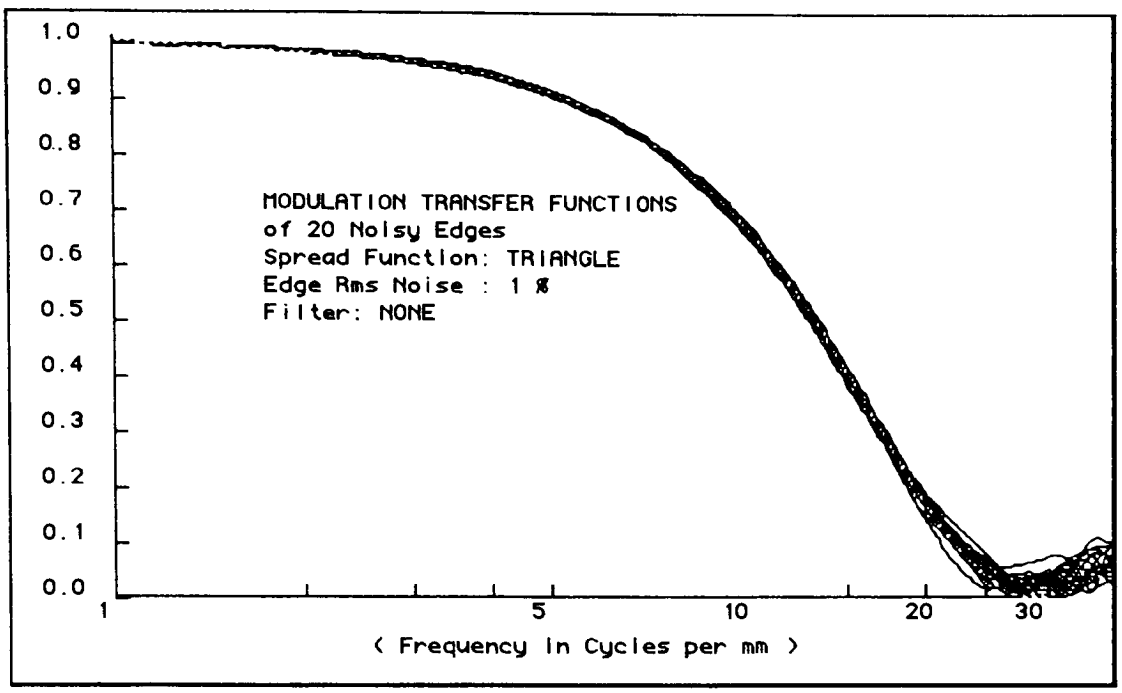


Figure AIII.4 99% confidence intervals for MTF RMS Differences. Filter treatments: none, damping (Dmp.), and chop (Chp.). 10% RMS Noise level.



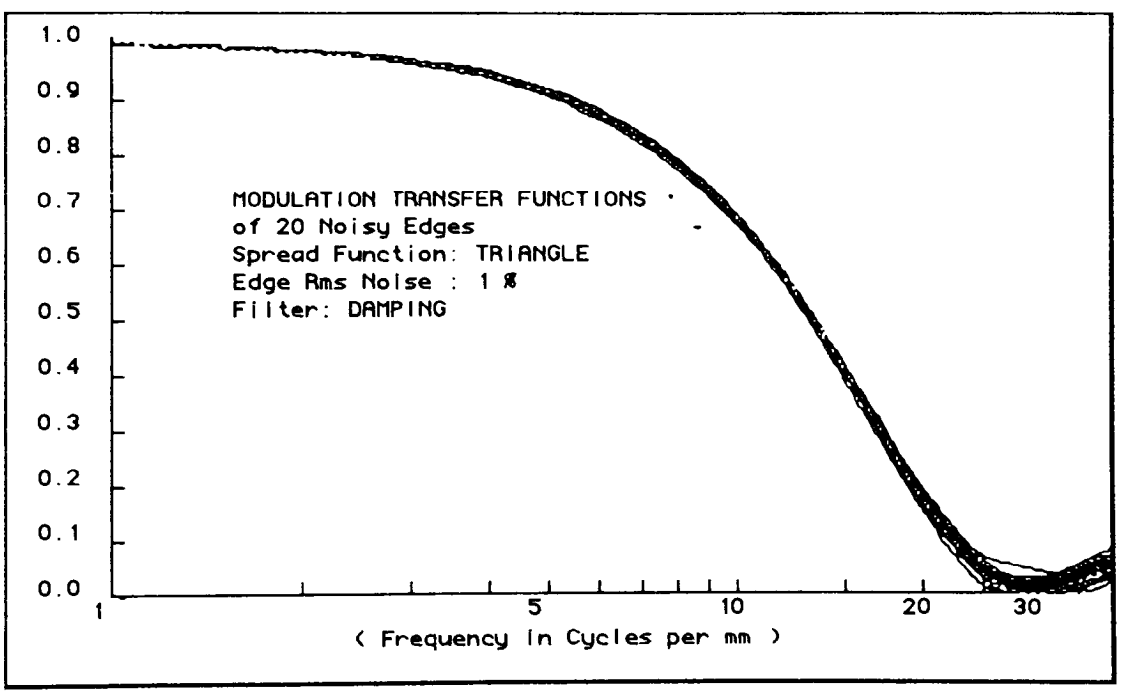


Figure AIII.5 Modulation Transfer Functions for 20 1% RMS noise edges generated from a triangle spread function. The edges of the two sets shown were processed with no filter & damping filter. A dotted line traces a noiseless reference MTF

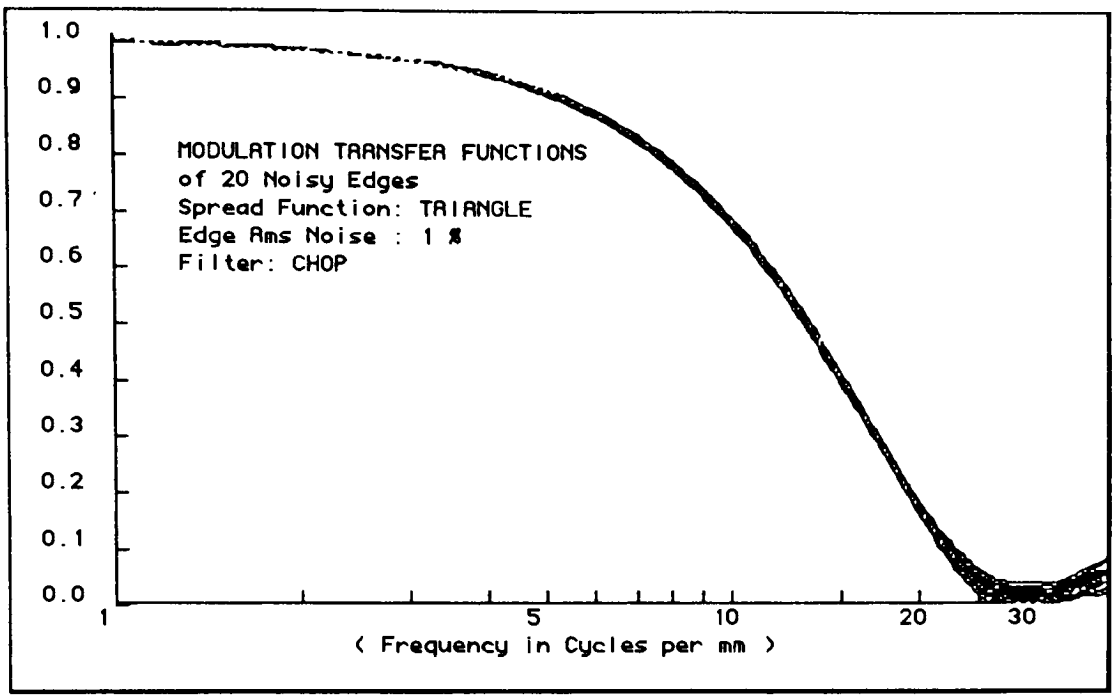


Figure AIII.6 Modulation Transfer Functions for 20 1% RMS noise edges generated from a triangle spread function with chop filter. A dotted line traces a noiseless reference MTF.

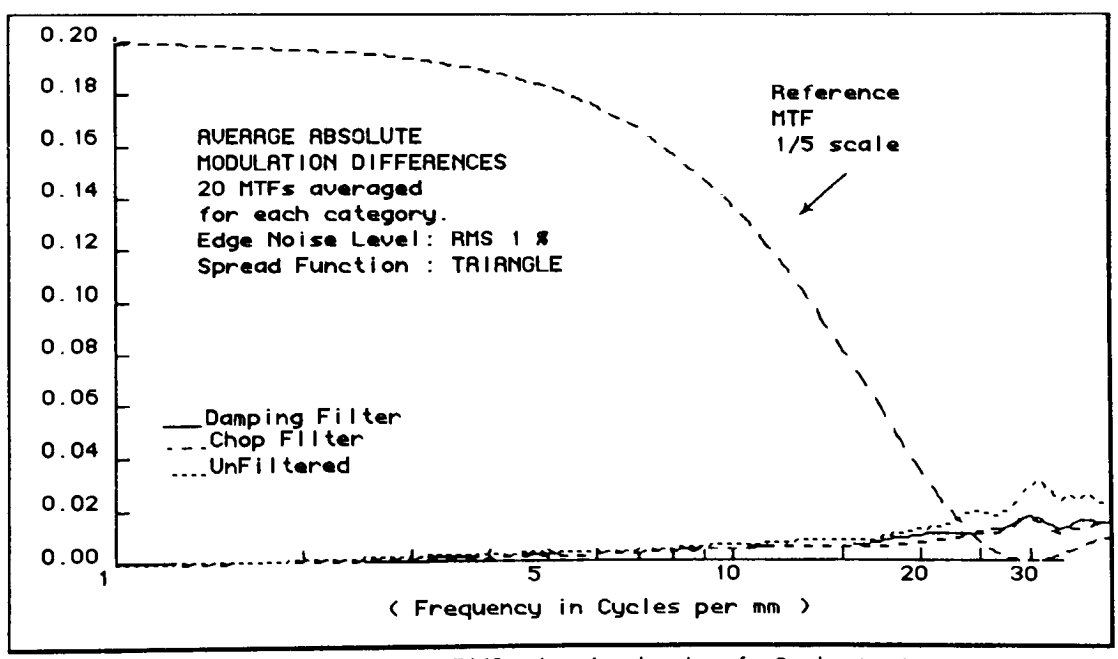
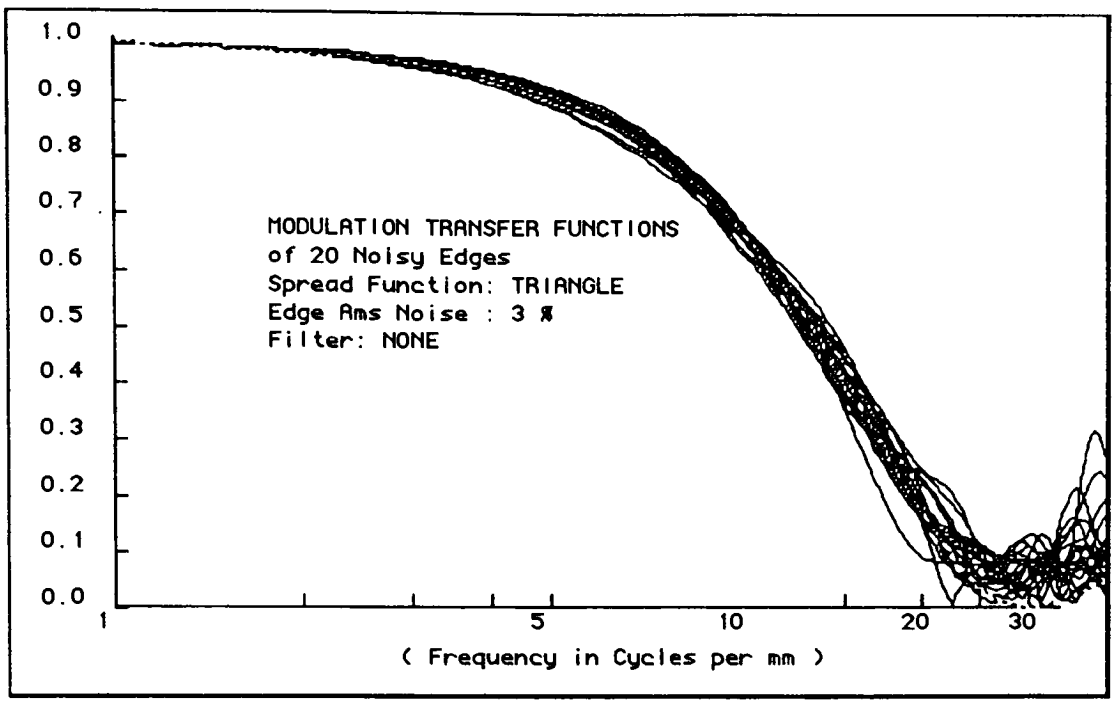


Figure AIII.7 Comparative errors in 1% RMS noise triangle edges for 3 edge treatments.



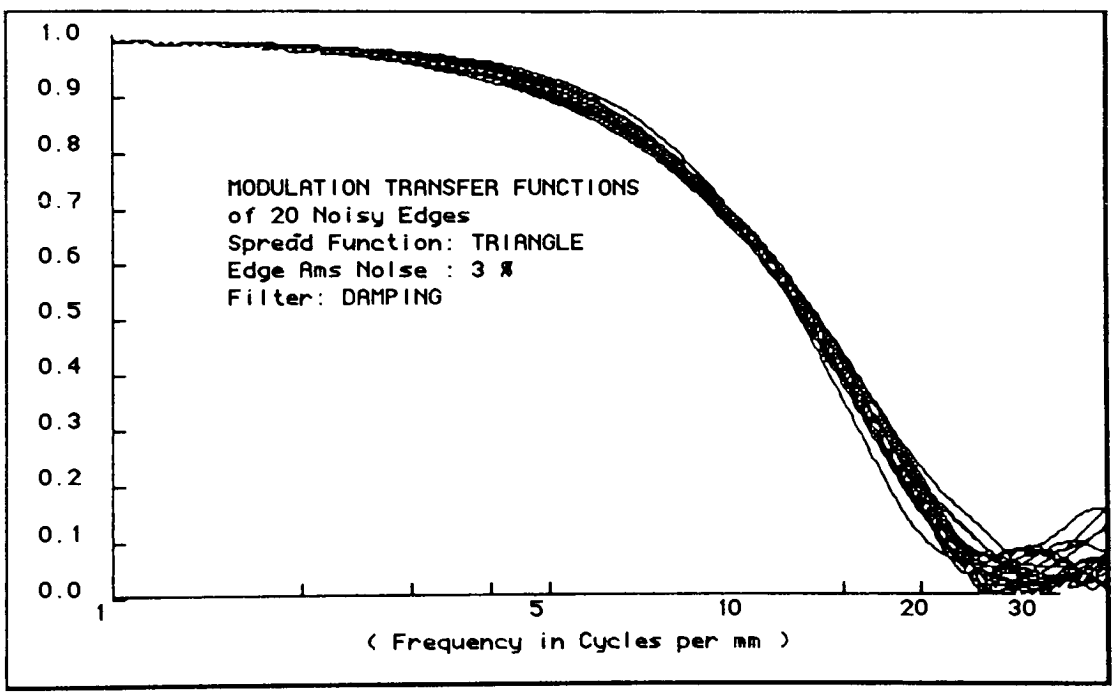


Figure AIII.8 Modulation Transfer Functions for 20 3% RMS noise edges generated from a triangle spread function. The edges of the two sets shown were processed as with no filter & Damping Filter. A dotted line traces a noiseless reference MTF.

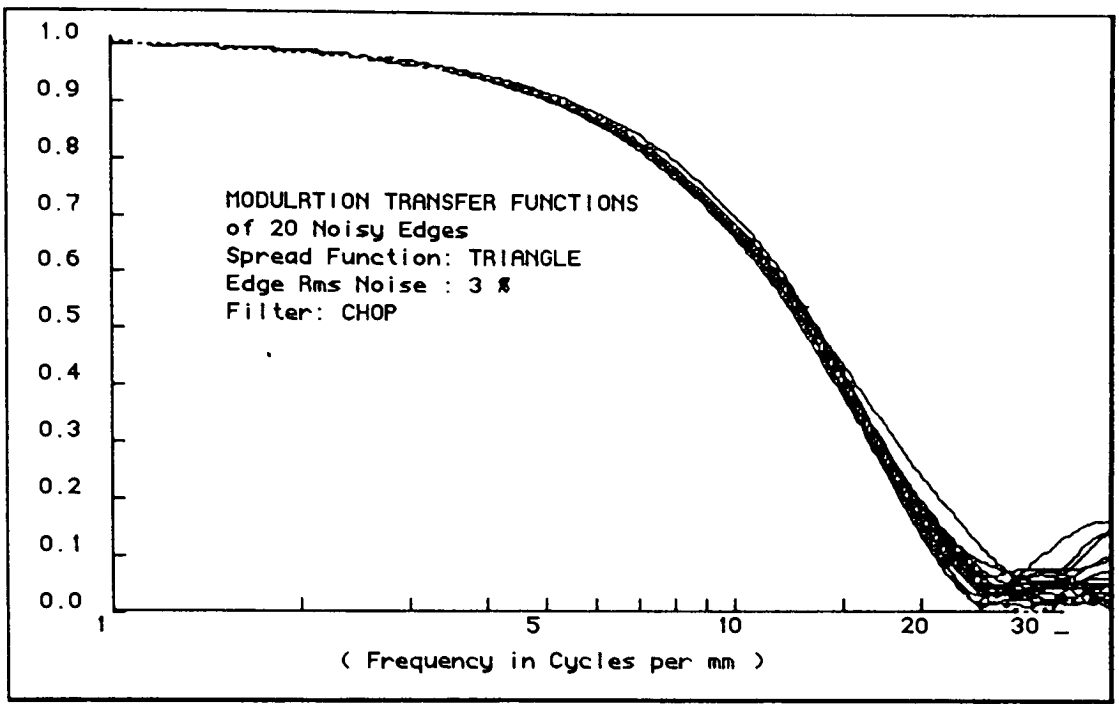


Figure AIII.9 Modulation Transfer Functions for 20 3% RMS noise edges generated from a triangle spread function with chop filter. A dotted line traces a noiseless reference MTF.

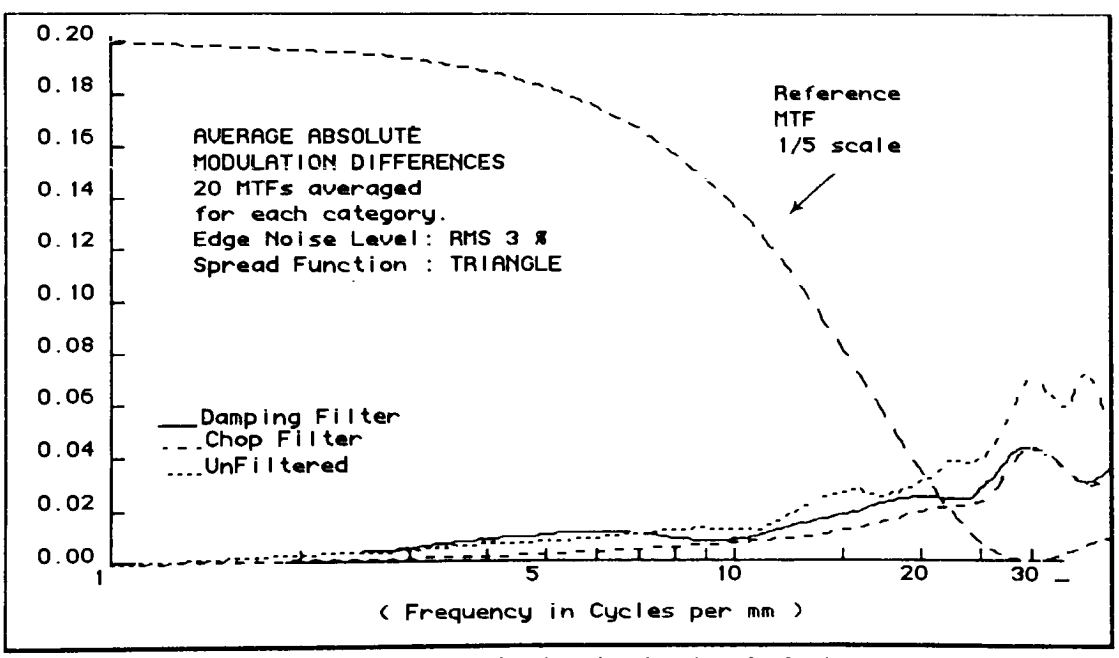
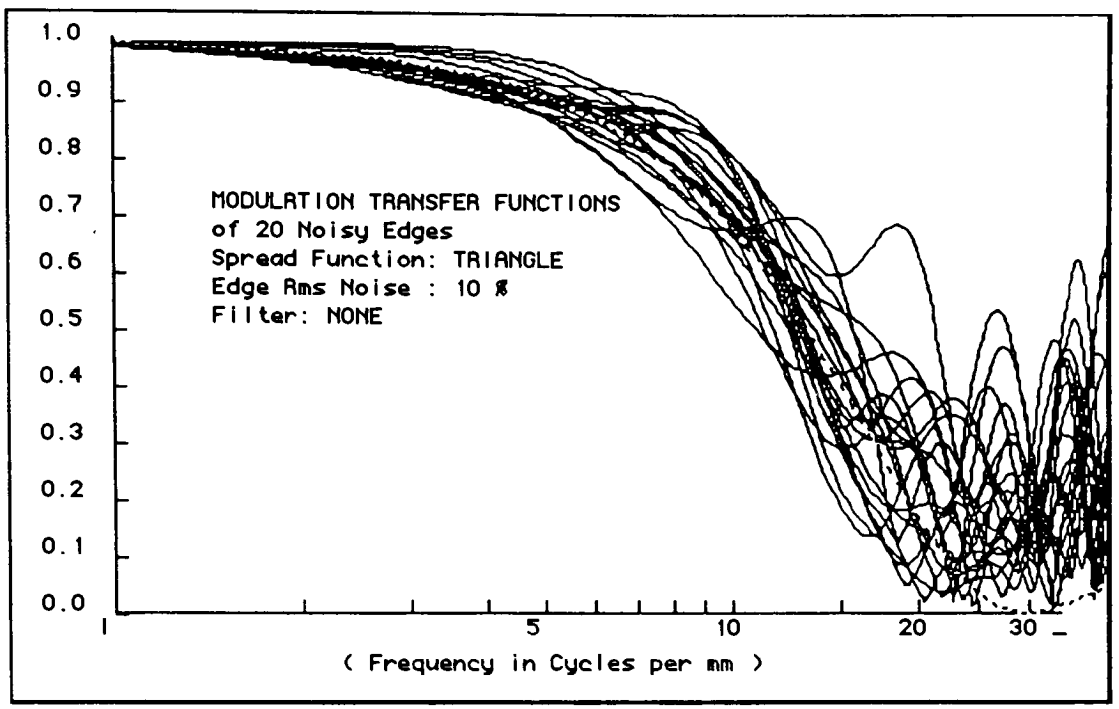


Figure AIII.10 Comparative errors in 3% RMS noise triangle edges for 3 edge treatments.



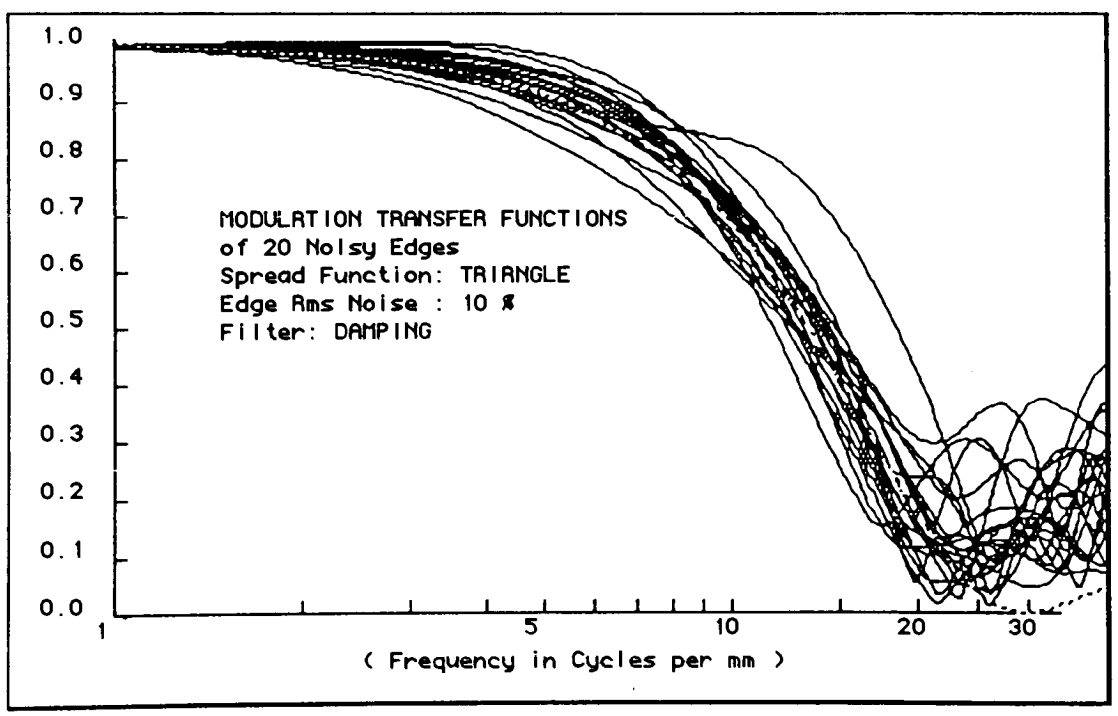
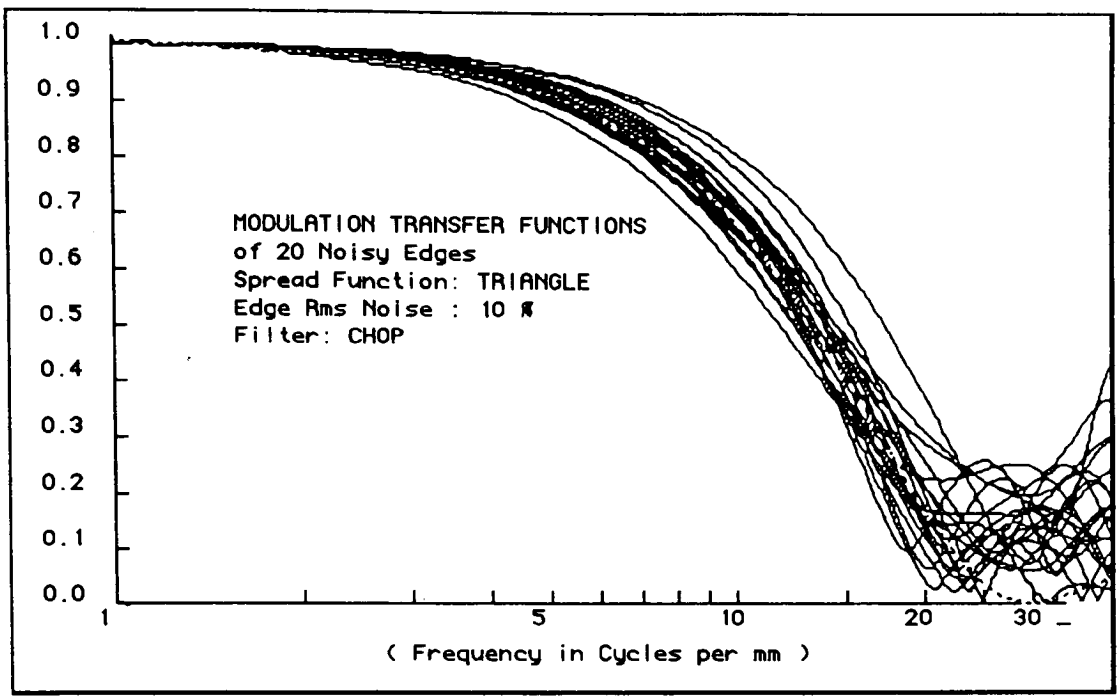


Figure AIII.11 Modulation Transfer Functions for 20 10% RMS noise edges generated from a triangle spread function. The edges of the two sets shown were processed with no filter & damping filter. A dotted line traces a noiseless reference MTF.



Figue AIII.12 Modulation Transfer Functions for 20 10% RMS noise edges generated from a triangle spread function with chop filter. A dotted line traces a noiseless reference MTF.

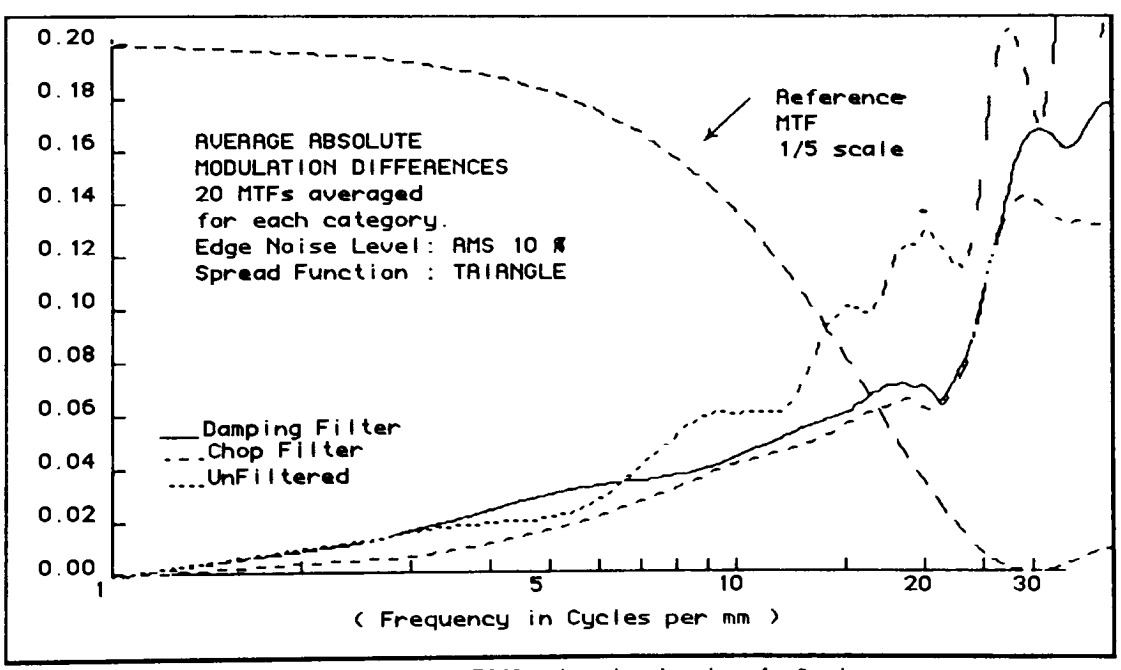
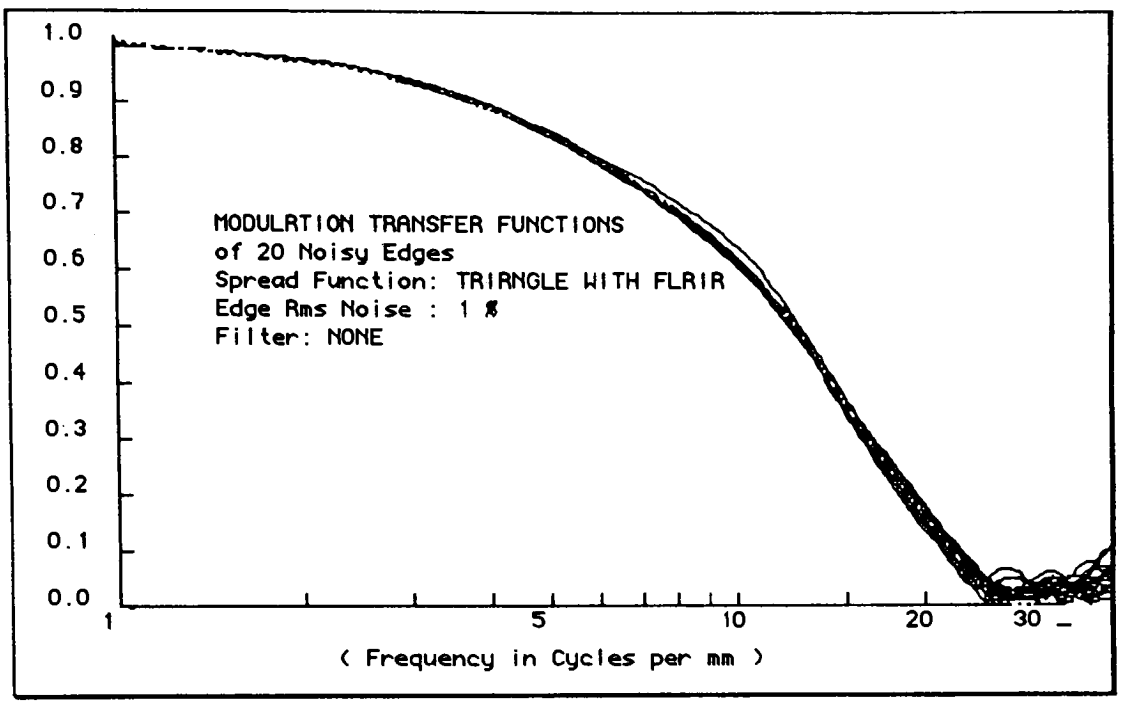


Figure AIII.13 Comparative errors in 10% RMS noise triangle edges for 3 edge treatments.



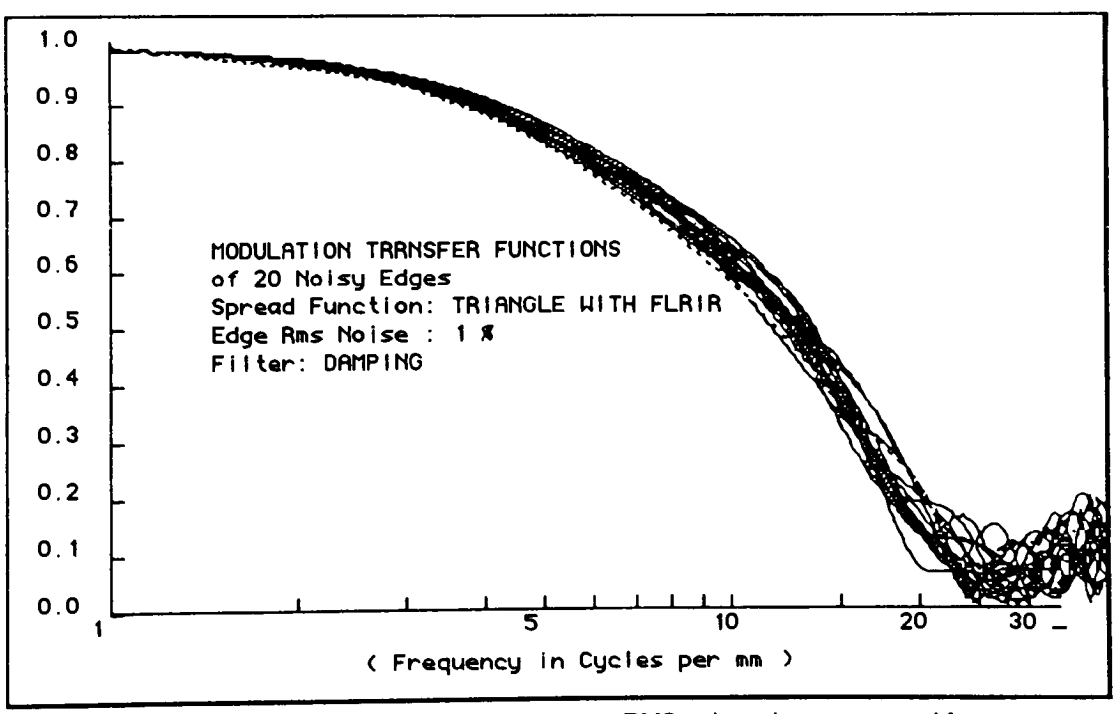


Figure AIII.14 Modulation Transfer Functions for 20 1% RMS noise edges generated from a triangle spread function. The edges of the two sets shown were processed with no filter & damping filter. A dotted line traces a noiseless reference MTF

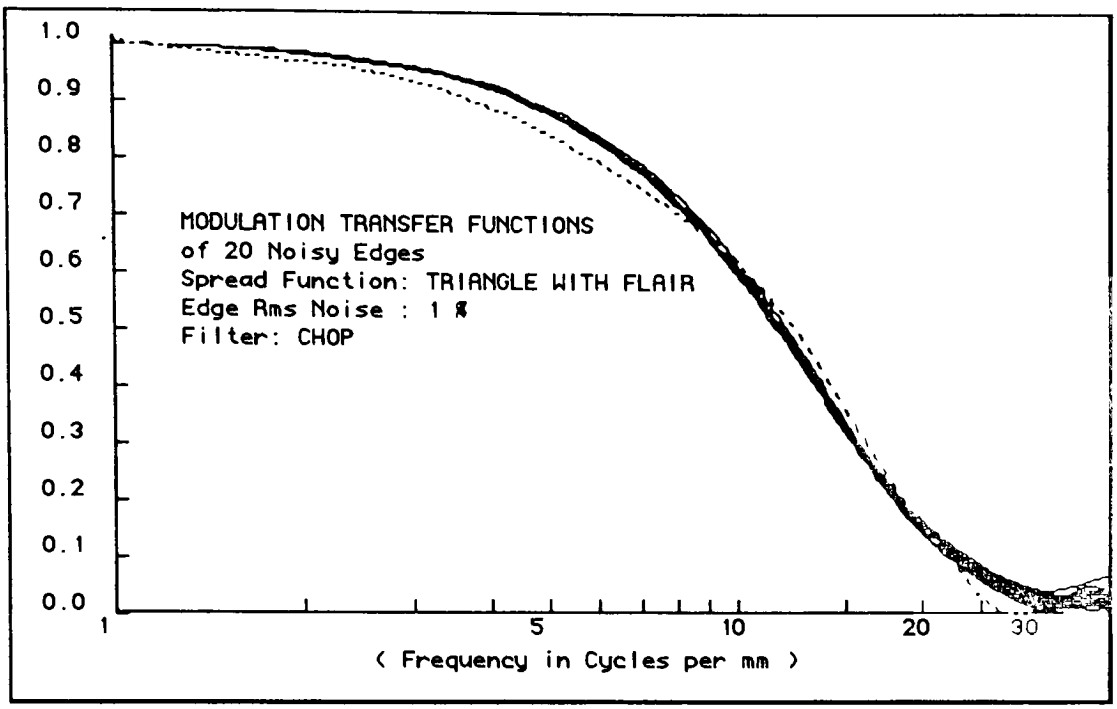


Figure AIII.15 Modulation Transfer Functions for 20 1% RMS noise edges generated from a triangle -flare spread function with chop filter. A dotted line traces a noiseless reference MTF. Note the overall displacement from the reference MTF.

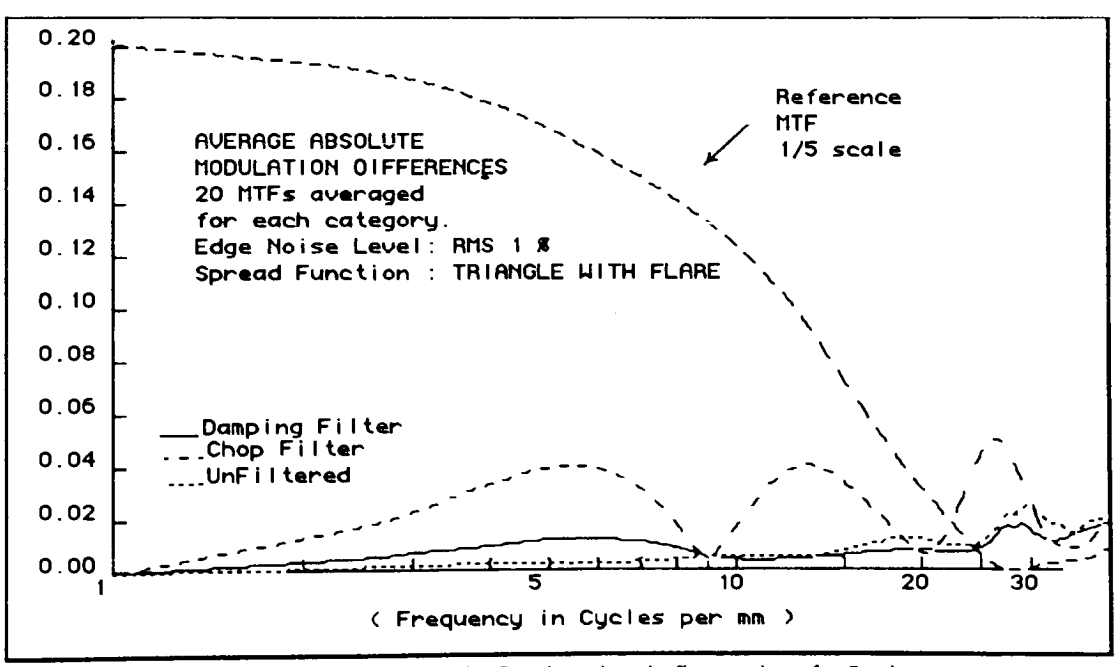
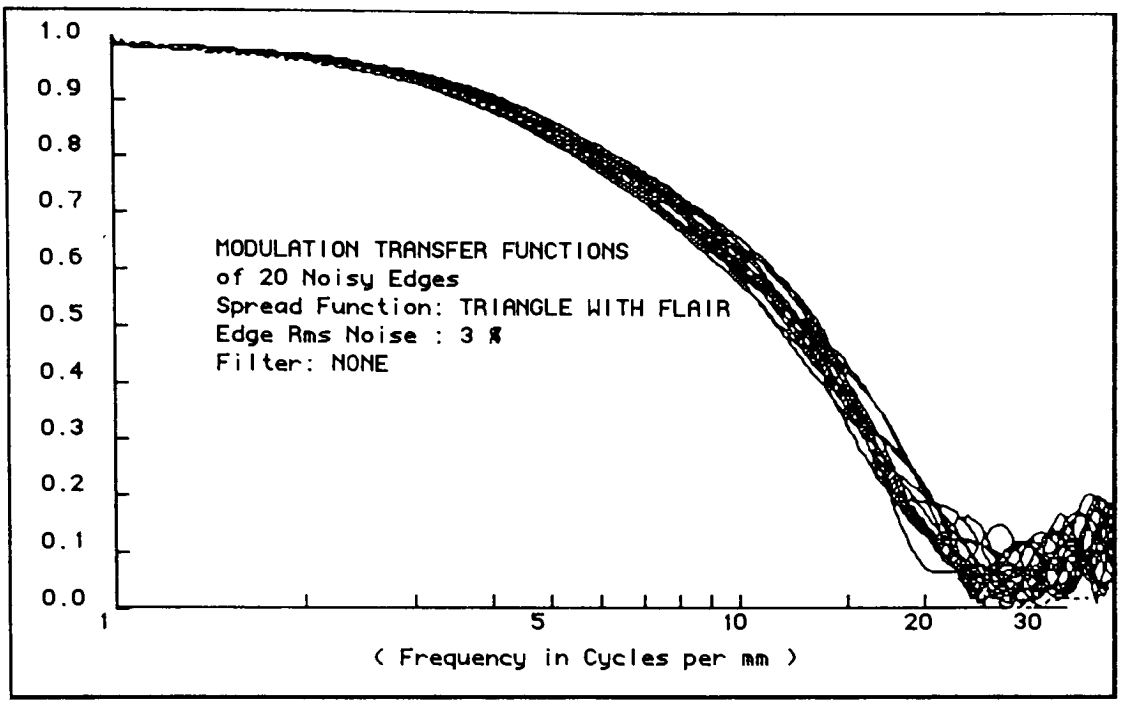


Figure AIII.16 Comparative errors in 1% RMS noise triangle-flare edges for 3 edge treatments.



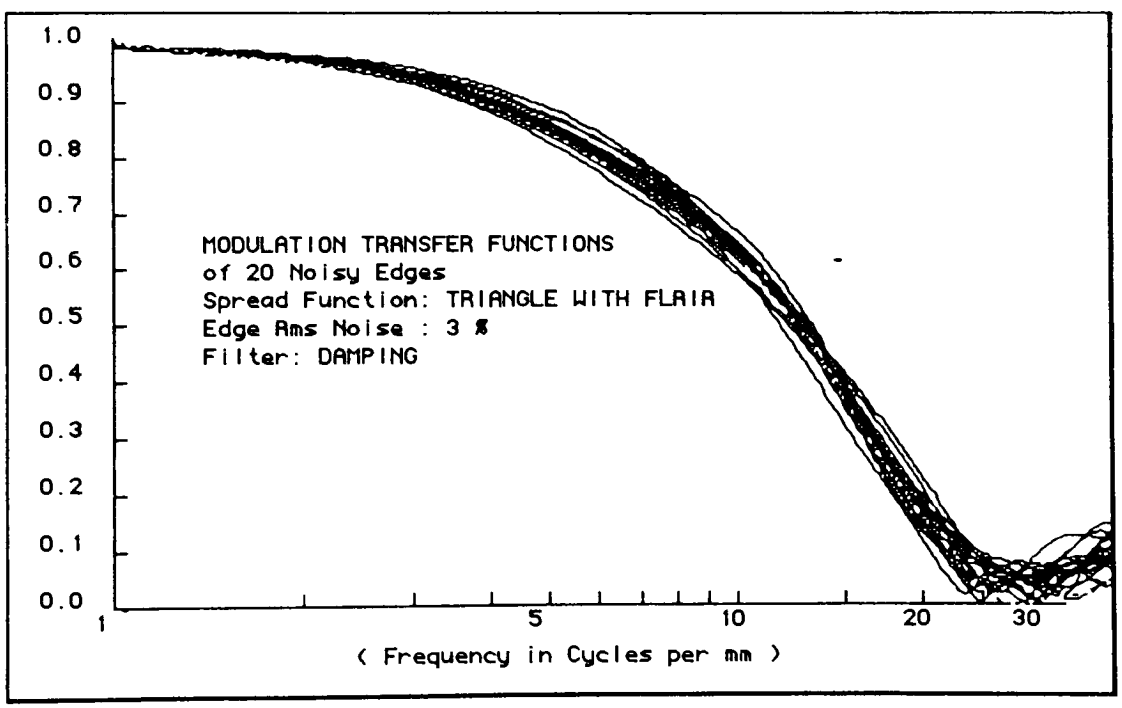


Figure AIII.17 Modulation Transfer Functions for 20 3% RMS noise edges generated from a triangle-flare spread function. The edges of the two sets shown were processed with no filter & damping filter. A dotted line traces a noiseless reference MTF.

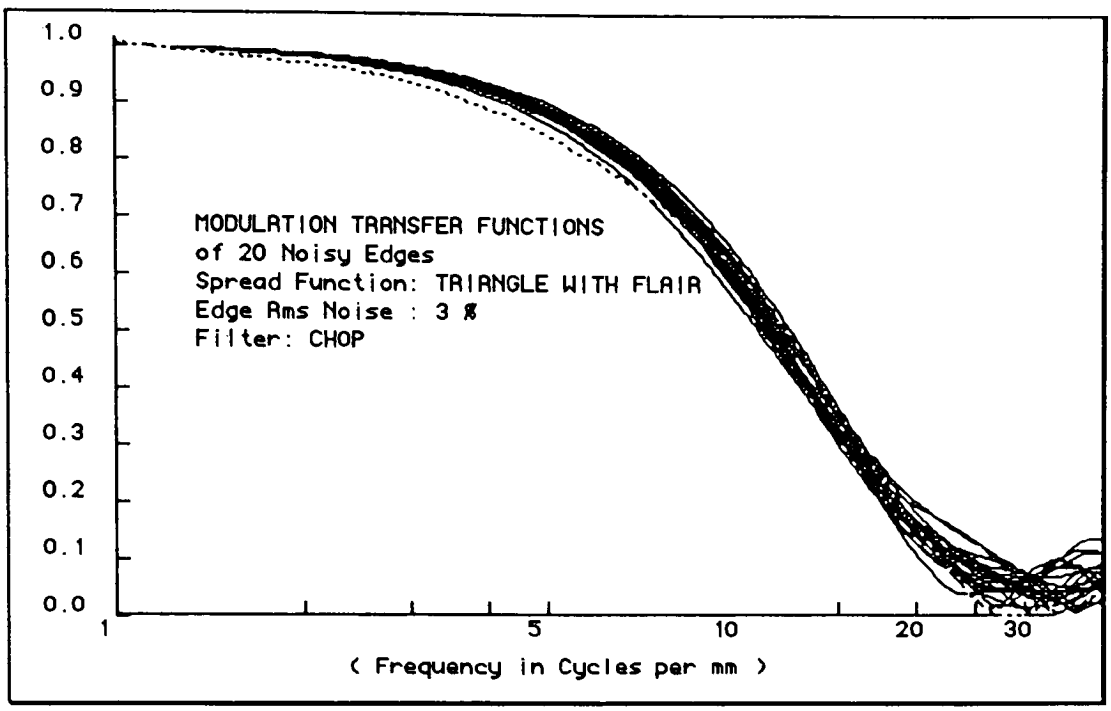


Figure AIII.18 Modulation Transfer Functions for 20 3% RMS noise edges generated from a triangle -flare spread function with chop filter. A dotted line traces a noiseless reference MTF.

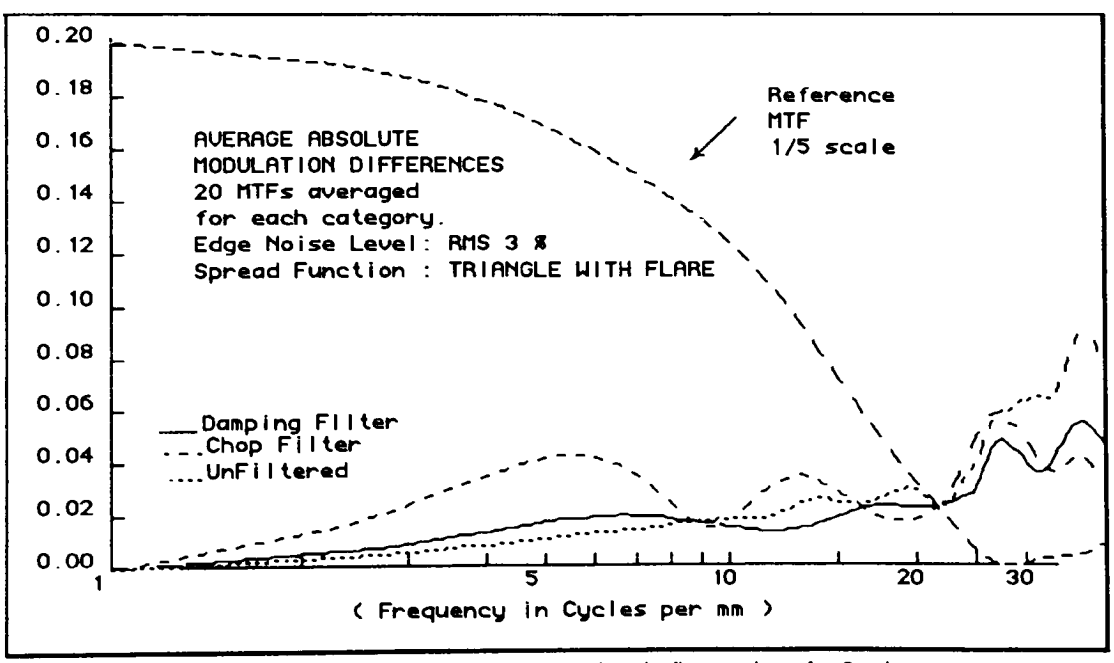
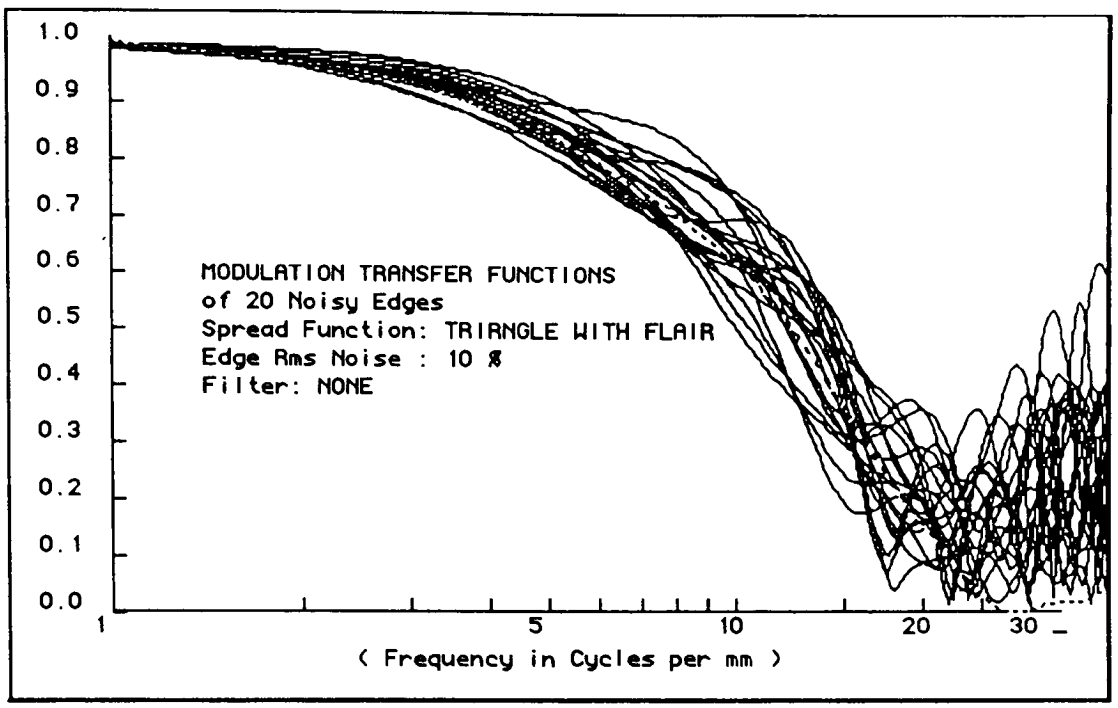


Figure AIII.19 Comparative errors in 3% RMS noise triangle-flare edges for 3 edge treatments.



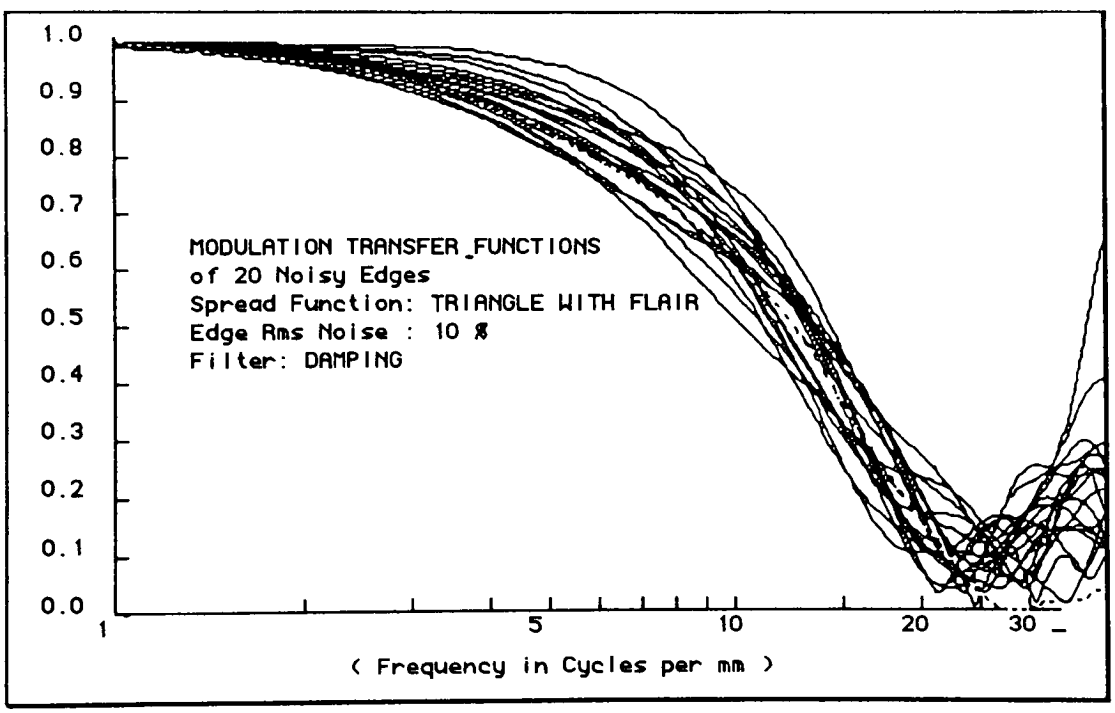


Figure AIII.20 Modulation Transfer Functions for 20 10% RMS noise edges generated from a triangle-flare spread function. The edges of the two sets shown were processed with no filter & damping filter. A dotted line traces a noiseless reference MTF.

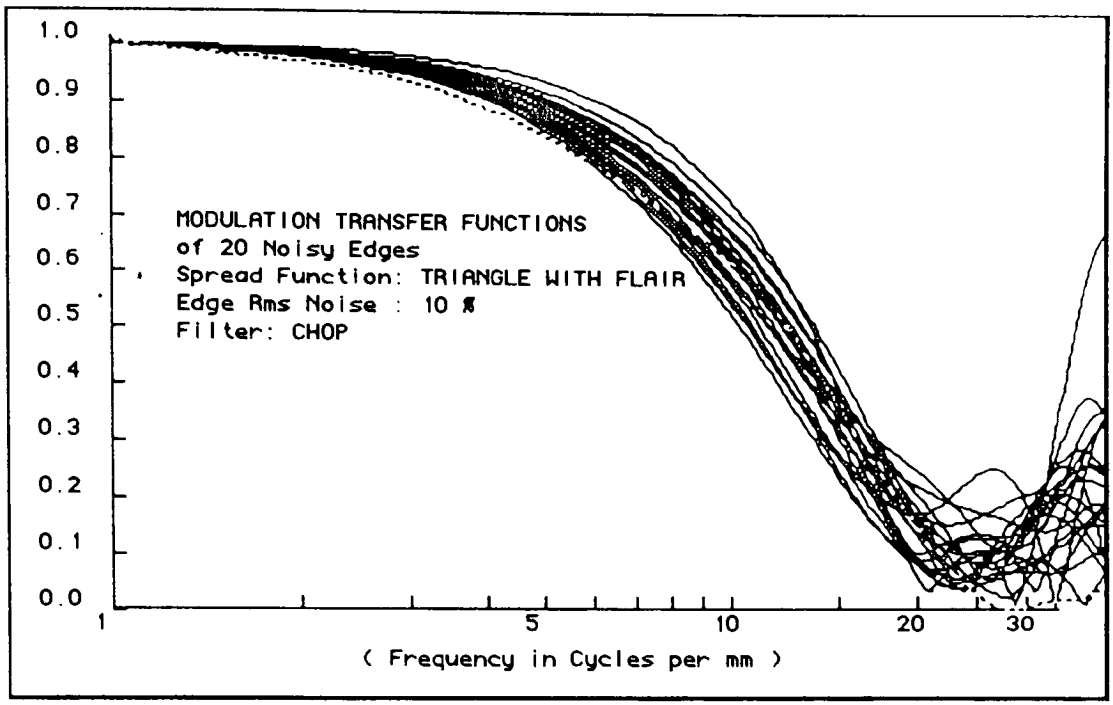


Figure AIII.21 Modulation Transfer Functions for 20 10% RMS noise edges generated from a triangle -flare spread function with chop filter. A dotted line traces a noiseless reference MTF.

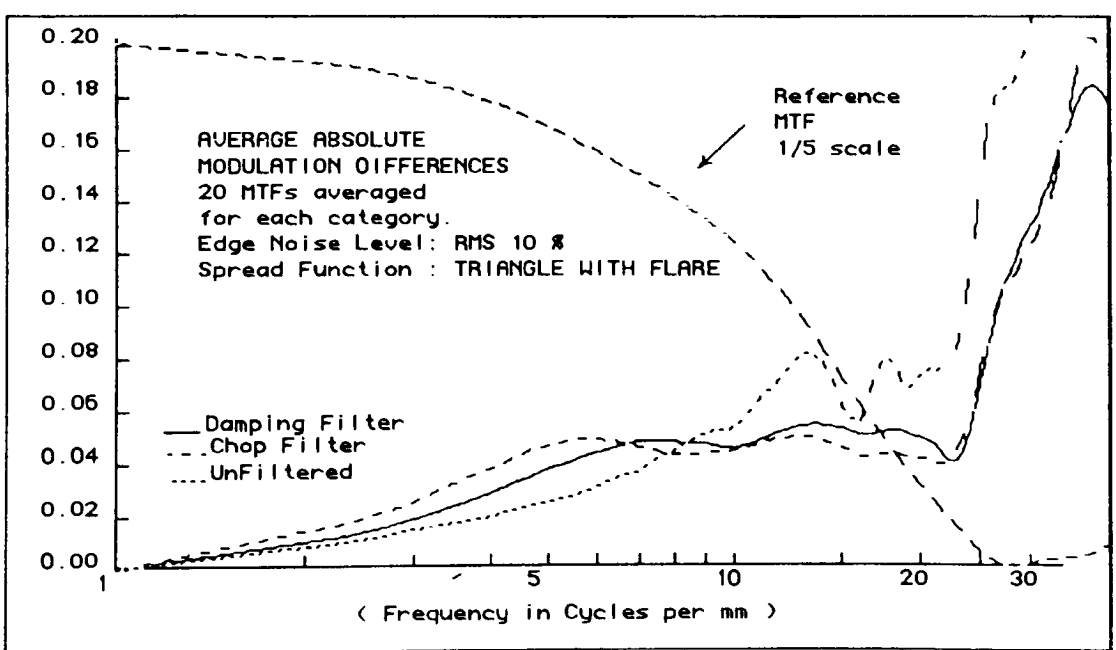


Figure AIII.22 Comparative errors in 10% RMS noise triangle-flare edges for 3 edge treatments.

APPENDIX IV

Source Code

Including Programs for:

Edge Generation

Edge to MTF Transformation

MTF Data Analysis

For assistance in reading
Forth source code
consult reference 15

SOURCE CODE INITIALIZATION

```
\ THESIS: a shift variant filter for edge trace analysis
\ David C. Johnson   NOV. 20, 1986
\ latest update 1/14/89

ONLY FORTH DEFINITIONS
ALSO MAC
ALSO SANE
FP
DECIMAL

\ ######################### Initialization ######################
1 CONSTANT Dmp           \ Filter labeling constants.
2 CONSTANT Chp
3 CONSTANT No
1 CONSTANT Gs            \ Spread function label constants.
2 CONSTANT GsPd          \              "
3 CONSTANT Tr            \              "
4 CONSTANT TrPd          \              "

VARIABLE ArrAdr          \ destination address for reading files
VARIABLE NFlag           \ edge pltng flag; 0=edge, 1=filtered edge
VARIABLE PlotFlag        \ plotting flag; 0=spatial, 1=frequency
VARIABLE Percent         \ Percent labeling variable
VARIABLE NToe            \ toe position on 256 point continuum
VARIABLE NShoulder       \ shoulder position
VARIABLE ToeStart        \ edge starting pt for MTF determination
VARIABLE ShoulderEnd     \ edge stopping pt for MTF determination
VARIABLE Type#           \ data type code; 1=edge, 2=filt edge, 3=MTF
VARIABLE Offset          \ offset for adjusting ArcSin Toe & shoulder
VARIABLE FltLbl          \ Filter mode graph label
VARIABLE Functlbl        \ Spread function graph label
VARIABLE Sprd            \ half spread (x) of edge
VARIABLE c               \ center of edge
VARIABLE Noise%          \ percent of noise for multiple
VARIABLE Typ             \
                         \ edge generators, 0 default.
FVARIABLE Sum            \ sum variable for integration of sprd funct
FVARIABLE SumDif         \ records sum of difs for comparing MTFs
FVARIABLE Top            \ mean of edge, upper plateau
FVARIABLE Bottom         \ mean of edge, lower plateau
FVARIABLE LgScl          \ Log scale variable for plotting

10 percent !             \ Default value
10 Noise% !              \ Default value
0 NFlag !                \ Default value
98 NToe !                \ Default value
158 NShoulder !          \ Default value
49 ToeStart !            \ Default value
207 ShoulderEnd !        \ Default value
1 Type# !                \ Default value
```

```
***** Created word commands for setting up arrays ******

: Incr4   ( AdrI - NewAdr )       \ Increments 4 bytes.
        4 * + ;

: Incr10   ( Adr I - NewAdr )     \ Increments 10 bytes.
        10 * + ;

: FPArray                 \ Sets up floating point arrays.
  CREATE 10 * ALLOT
  DOES> SWAP Incr10 ;

: Array                   \ Sets up integer arrays.
  CREATE 4 * ALLOT
  DOES> SWAP Incr4 ;

21   Array  Pt-Arr

260   FPArray Data-Arr      \ Floating Pt. arrays
260   FPArray DataI-Arr     \ Preceding number indicates
260   FPArray DataII-Arr    \ number of items in array.
260   FPArray DataIII-Arr
260   FPArray DataIV-Arr
260   FPArray FlData-Arr
260   FPArray Temp-Arr
260   FPArray Mtf-Arr
120   FPArray Abs-Arr
50    FPArray RMS-Arr
20    FPArray c-Arr
20    FPArray sprd-Arr

\ ******************  MTF  Section Initialization ******************

 FVARIABLE  Frequency          \ F
 FVARIABLE  Increment          \ E
 FVARIABLE  FreqIncr           \ (1/2E)/80000
 FVARIABLE  PiFE2              \ PiFE2
 FVARIABLE Llimit              \ lower limit of modulation of std
                               \ being considered for comparison.


******************  Default settings  ****************************

 .01667 Increment F!           \ default sampling Increment in mm
.01     Llimit F!              \ default
 0      Offset !               \ default
```

```
\ ********* Created words for initializing OTF Program ************

: PiE2  ( - n )                  \ Returns Pi * Increment * 2.
        3.1415926 Increment F@
        F* 2. F* ;

: FreqScale     ( - )            \ Sets Frequency Increment, based on the
        1.                       \ sampling Increment, as the variable,
        Increment F@ 2. F*       \ FreqIncr
        F/
        .01 F*
        FreqIncr F! ;
```

```
\ ################### Gaussian Noise Generator  ############

CODE  Randomize                     \ Resets Macintosh random
        MOVE.L (A5),A0              \ number generator.
        MOVE.L $20C,-126(A0)
        RTS
END-CODE

: Noise { prcnt | temp - Fn }       \ Noise generator 0 to 1.000
        FP                          \ Gaussian dist. about 0.
        0 -> temp                   \ Expects percent on
        12 0                        \ parameter stack.
        DO
           CALL Random ABS          \ Returns a random number
           1001 MOD                 \ from -1.000 to 1.000
           +> temp                  \ Ensures that center of noise
        LOOP                        \ is at 0. To get a gaussian
           temp                     \ distribution 10
           I>F 1000. F/             \ samples of noise are averaged.
           6. F-                    \ Division by 1000 yields
           prcnt I>F                \ Nos. from -1.000 to 1.000.
           100. F/ F* ;

\ @@@@@@@@@@@@@@@@ Spread Function  Generators @@@@@@@@@@@@@@

\ ###################### Triangle  ############################

: TriFunct  { npp | - }             \ Tri  b = 1, Yo = 1,000,000
        256 0 DO                    \ Expects n% on parameter stack
        0. i Temp-Arr  F!           \ which is % of a
        LOOP                        \ normalized function.
        129 101 DO
           I  100 - I>F
           28. F/ FDUP
           npp I>F 100. F/
           F*
           I Temp-Arr  F!
           1. FSWAP F-
           npp I>F 100. F/
           F*
           I 28 + Temp-Arr F!
        LOOP
        -1 PlotFlag ! ;
```

```
\ ######################  Gaussian  ########################

: GaussF        { nb npp | - }  \ Produces the Gaussian function
        256 0                   \ y = 1/b * e^-(x/b)^2
        DO                      \ The scaling factor, b,
           i 128 -              \  determines height and breadth.
           I>F nb I>F F/        \ Expects scaling factor (nb)
           2. Fy^x FNEGATE      \ and percent (npp) on the
           256.  F/             \ parameter stack.
           Fe^x nb I>F F/
           npp I>F 100. F/
           F*
           i Temp-Arr F!
        LOOP ;

: TransferFiles ( adr1 adr2 - ) \ Transfers data from one FP
        SWAP                    \ array to another. Expects
        256 0                   \ the beginning address of
        DO                      \ the source array followed
            i 10 * + F@         \ by the beginning address
            i 10 * + F!         \ of the receiving array.
        LOOP ;

: TransferTD    ( - )           \ Transfers data from
        256 0                   \ Temp-Arr to data-Arr.
        DO
            i Temp-Arr F@
            i Data-Arr F!
        LOOP ;

: AddFilesTD            ( - )
        256 0                   \ Adds Temp-Arr to Data-Arr
        DO
           i Data-Arr F@
           i Temp-Arr F@
           F+
           i Data-Arr F!
        LOOP ;


\ ################  Gaussian with Flare  ######################

: GaussPed      ( - )           \ Produces the function,
        1 90 GaussF             \ 0.9Gauss1 + 0.1gauss2 where
        TransferTD              \ 0.9Gauss1 is 90% of a
        4 10 GaussF             \ normalized gaussian function
        AddFilesTD ;            \ and 0.1Gauss2 is 10% of a 1/b
                                \ gauss function and stores
                                \ it in Data-Arr.
```

```
\ ##################  Triangle With Flare  ######################

: TriPed        ( - )           \ Produces the function,
        90 TriFunct             \ 0.9Tri + 0.1gauss2 where
        TransferTD              \ 0.9Tri is 90% of a Triangle
        4 10 GaussF             \ function and 0.1Gauss2 is
        AddFilesTD ;            \ 10% of a 1/b  gauss function,
                                \ and stores it in Data-Arr.


\ ###################### Edge Generator ##########################

: Edge ( - )                    \ Accumulating numerical
                                \ integration of a function
        0.  Sum F!              \ with dx = 1. Expects an
        256 0 DO                \ existing  Data-Arr and
           I  Data-Arr  F@      \ converts the spread function
           Sum F@ F+  SUM F!    \ data to edge data. Scales all
           Sum F@ I Data-Arr F! \ edge array values to 1.
        LOOP
        256 0 DO
           I Data-Arr F@ SUM F@  F/
           I Data-Arr F!
        LOOP
        -1 PlotFlag !
        1 Type# ! ;


: NoiseEdge { p | temp - }  \ Converts Data-Arr from Edge
        Randomize           \ to Noise Edge values.
        256 0 DO            \ Reseeds random number generator
           I Data-Arr  F@   \ Adds a Gaussian noise value to
           p Noise F+       \ each array edge value.
           I Data-Arr F!    \ Expects % on parameter stack
        LOOP
        -1 PlotFlag ! ;
: TriEdN        { pp | - }  \ Generates an edge based on a
        100 Trifunct        \ a triangle spread function and
        TransferTD
        Edge                \ having n % noise.
        pp NoiseEdge ;      \ Expects n% on parameter stack


: TriPdEdN { pp | - }       \ Generates an edge based on a
        TriPed              \ 90% triangle spread function
        Edge                \ on a 10% 1/4 gaussian base.
        pp NoiseEdge ;      \ Expects n% on parameter stack.
```

```
: GsEdN { pp | }            \ Generates an edge based on
       1 100 GaussF         \  a 1/1 gaussian function.
       TransferTD
       Edge
       pp NoiseEdge ;

: GsPdEdN  { pp | }      \ Generates an edge based on a
       GaussPed          \ 90% gaussian 1/1 function on a
       Edge              \ 10% gaussian 1/4 function flare.
       pp NoiseEdge ;

: GsEdgs  ( - )              \ Generates 20 Gaussian noise
       20 0                  \ edges. Set % by storing
       DO                    \ whole % in "Noise%. "Creates
           i 1+ . CR         \ file which should be named
           Noise% @ GsEdN    \ GsEdg%N# percent and # must
           MakeFile          \ be included in name.
       LOOP ;

: GsPdEdgs  ( - )               \ Generates 20 Gauss flare
       20 0                     \ noise edges Set % by storing
       DO                       \ whole % in "Noise%" Creates
           i 1+ . CR            \ file which should be named
           Noise% @ GsPdEdN     \ GsPdEdg%N# percent and # must
           MakeFile             \ be included in name.
       LOOP ;

: TriEdgs  ( - )              \ Generates 20 Triangle noise
       20 0                   \ edges. Set % by storing
       DO                     \ whole % in "Noise%" Creates
           i 1+ . CR          \ file which should be named
           Noise% @ TriEdN    \ TriEdg%N# percent and # must
           MakeFile           \ be included in name.
       LOOP ;

: TriPdEdgs  ( - )              \ Generates 20 Triangle
       20 0                     \ flare noise edges.
       DO                       \ Set % by storing whole %
           i 1+ . CR            \ in "Noise%". Creates file
           Noise% @ TriPdEdN    \ which should be named
           MakeFile             \ TriPdEdg%N# percent and #
       LOOP ;                   \ must be included in name.
```

EDGE FILTERING ROUTINES

```
\ @@@@@@@@@@@@@@@  PREPARATION FOR EDGE FILTERING  @@@@@@@@@@@@@@@@@@

VARIABLE Width      \ Number of points in averaging block in AverageN
VARIABLE Index      \ Relative position of center of block
VARIABLE Direction  \ Is -1  for left or +1 for right
VARIABLE Start      \ Location for starting Locate__
VARIABLE ni         \ integer version of index number
VARIABLE Jump       \ offset for jumping to shoulder (0 or 176)

VARIABLE SpanEdg    \ Extent of the edge on x axis, always an odd #
VARIABLE HalfSpan   \ Half of SpanEdg
VARIABLE Center     \ Center point of edge on x axis.
VARIABLE Sprd       \ Amount of error between toe and shoulder
VARIABLE Auto       \ variable to turn on automatic toe/Shld locator

FVARIABLE nn        \ floating point version of index number
FVARIABLE Sumx      \ sum of the values of x in sample
FVARIABLE SumDif2   \ sum of the deviations from mean squared
FVARIABLE Mean      \ mean of signal data
FVARIABLE StDev     \ Standard deviation of signal data
FVARIABLE ToeMean   \ Local mean of toe plateau
FVARIABLE ShldMean  \ Local mean of Shoulder plateau
FVARIABLE EdgScale  \ Scaling factor for normalizing noisy edge data

FVARIABLE AveN          \ Current average of n datapoints
FVARIABLE X             \ Last value of x
FVARIABLE Sum2          \ Summing variable

260 FPArray  Temp-Arr   \ Temporary array for holding edge data

60 Width !              \ Default width of averaging block  -
78 ni !                 \ # of pts used to determine local mean
1 Direction !           \ Default direction used in Locate__
1 Auto !                \ Default AutoPts selection

\ ############# Toe/Shoulder Location  ####################

: AutoPts       ( - )           \ Activates automatic
        1 Auto ! ;              \ estimation of toe
                                \ and shoulder points.
: SelectPts     ( ctr sprd - )
        0 Auto !                \ Allows selection of center and
        Sprd !                  \ edge spread for edge filtering
        Center ! ;              \ reference points.
```

```
: AverageN  ( - FAve )          \ Averages N consecutive data points.
        1 Direction !
        0. Sum F!                 \ Expects that Data-Arr exists
        Width @ Direction @ * 0   \ and that a starting position
        DO                        \ and direction; -1 or +1, have been
           Start @                \ stored in "Start"
           I Width @ 2 /          \ and "Direction" respectively
           Direction @ * -        \ and that the current position
           Index @ +              \ offset, index, has
           +                      \ been updated.
           Data-Arr
           F@
           Sum F@ F+
           Sum F!
        Direction @
        +LOOP
        Sum F@  Width @ I>F  F/
        FDUP AveN F! ;

: AveragePts {ind | hlf wd - Fnave}      \ Averages N consecutive
        30 DUP -> wd 2 / -> hlf          \ data points. Expects
        0. Sum2 F!                       \ that Data-Arr exists
        wd                               \ and that the current
         0                               \ position offset,ind,
        DO                               \ is on the parameter stack.
           ind
           hlf -
           i +
           Data-Arr F@
           Sum2 F@ F+ Sum2 F!
        LOOP
        Sum2 F@ wd I>F  F/ ;

: MeanCalcTS    ( - n )
        ni @ I>F nn F!    \ Finds mean for ni points at beginning
        0. Sumx F!        \ of array; pre toe direction = 1, or
        Direction @ -1    \ at end of array; plateau direction =-1.
        =
        IF
           176 Jump !
        ELSE
           0 Jump !
        THEN
        ni @  0
        DO
           I Jump @ +
           Data-Arr F@
           Sumx F@ F+ Sumx F!
        LOOP
        Sumx F@ nn F@ F/ ;
```

```
: ToeStat       ( - )           \ Sets variable, ToeMean
        1 Direction !           \ for current noisy edge.
        MeanCalcTS ToeMean F! ;

: ShldStat      ( - )           \ Sets variable, ShldMean
        -1 Direction !          \ for current noisy edge.
        MeanCalcTS ShldMean F! ;

: ScaleCalc     ( n1 n2 - )     \ Sets variable EdgScale
        F-                      \ for normalizing edge data.
        1. FSwap F/             \ Expects ShldMean &
        EdgScale F! ;           \ ToeMean on FP stack.

: NormEdge      ( - )           \ Normalizes any edge data
        ToeStat                 \ in Data-Arrfrom low mean
        ShldStat                \ to high mean of 0.0 to 1.0.
        ShldMean F@
        ToeMean F@
        ScaleCalc
        256 0
        DO
           I Data-Arr F@
           ToeMean F@ F-
           EdgScale F@ F*
           I Data-Arr F!
        LOOP ;

: FSin-1   ( nr - na )  \ Returns the angle having
        FDUP            \ nr as sine (radians).
        2. Fy^x         \ Expects a sine on the FP stack.
        1. FSWAP
        F- FSQRT
        F/
        FTan-1 ;
```

```
: FindEndPts     { Strt stp | nn# n# - }
        1 -> n#
        Strt -> nn#                 \ Locates 10 matched points
          BEGIN                     \ within the edge between
            nn# AveragePts          \ the starting point and
            .03 n# I>F F* .1 F+     \ stopping point.  Expects
            F>                      \ numbers on the stack.
            IF
               nn#
               n# Pt-Arr !
               1 +> n#
            THEN
            1 +> nn#
            n# 11 =
          UNTIL
        Stp -> nn#
        1 -> n#
          BEGIN
            nn# AveragePts
            .03 n# I>F F* .1 F+
            1. FSWAP F-
            F<
            IF
                nn#
                n# 10 + Pt-Arr !
                1 +> n#
            THEN
            -1 +> nn#
            n# 11 =
          UNTIL ;

: FindcenterAve { | sum# - n }   \ Uses matched point method
        0 -> sum#                \ to locate center of edge.
        11 1                     \ puts that integer on stack.
        DO
            i 10 + Pt-Arr @
            i Pt-Arr @
            -
            2 /
            i Pt-Arr @ +
            +> sum#
        LOOP
        sum# 100 * 50 + 1000 /  1+
        ." c = " DUP . CR ;

: LocateP        ( - n )         \ Locates the center point
        80 176 FindEndPts        \  of the edge.
        1 Pt-Arr @
        FindCenterAve ;
```

```
: ArcSin   { t c | hw -  }       \ Uses ArcSin method to estimate
              0. Sum F!          \ toe & shoulder. Expects
        c t - DUP -> hw 2 *      \ approximate "10%-from-toe-point"
        0                        \ and center.For each x value the
        DO                       \ constant k is calculated where:
            i t + AveragePts     \ k = [Arc(Sin2(y-.5))]/x
            .5 F- 2. F*          \ Absolute k values are
            FSin-1               \ summed and averaged to obtain an
            i hw -               \ estimate of K for the sine function.
            0 <>
            IF
                i hw - I>F F/ FABS
                Sum F@ F+ Sum F!
            ELSE
                FDROP
            THEN
        LOOP
         SUM F@
         hw 2 * 1 - I>F          \ Divides k sum by [width -1] ie.
         F/                      \  # of k's summed.
         2. F*
         Pi FSWAP F/
         .5 F+ F>I DUP
         c +  Offset @ + NShoulder  !
         c SWAP - Offset @ - NToe ! ;

: LocateN        ( -  )  \ Estimates the toe and shoulder
        NormEdge         \ points of an edge function if
        Auto @           \ AutoPts is activated or allows
        IF               \ operator selection of center point
            LocateP      \ and spread if SelectPts is activated.
            ArcSin
        ELSE
              Center @ 128 + DUP  \ Select option section
              Sprd @ 32 + DUP
              ROT +
              NShoulder !
              - NToe !
              ." c = " Center @ .
              " Sprd = " Sprd @ .
              ." Toe = " NToe @ .
              ." Shoulder = " NShoulder @ .
        THEN ;
```

```
: SetPlateau     { pt dr | - }    \ Sets toe plateau to 0 if
        dr 1 +                    \  Direction is 1 or sets
        IF                        \ shoulder plateau to 1 if
           pt                     \ direction is -1. Expects
           0                      \ Ntoe or Nshoulder and
           DO                     \ direction on parameter stack.
              0. I Data-Arr F!
           LOOP
        ELSE
           256
           pt
           DO
              1. I Data-Arr F!
           LOOP
        Then ;
```

```
\ @@@@@@@@@@@@@@ SHIFT  VARIANT  FILTER  ROUTINES  @@@@@@@@@@@@@@@

\  USES THIS OPERATION ON PLATEAUS OF EDGE INFORMATION:

\          E(m)   =  summation of e(i) from N to ( 2Q - 1 + N )
\                    ----------------------------------------
\                                ( 2Q - 1 )
\
\        Where   E(m) = Filtered Edge Function
\                e(i) = edge function with noise
\                N = X position of shoulder or toe
\                Q = ( M - N )
\                M = variable X position on plateau of edge function

\ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

: SumAve  { N | SumSpan - n }    \ Gives the average over "SumSpan"
        0.  Sum F!               \ I is equivalent to Q in equation.
        I  2 * 1 - -> SumSpan    \ Expects NToe or NShoulder (value).
        Sumspan 1 +
        1
        DO

           N  I Direction @ * -
           Data-Arr F@
           Sum F@ F+ Sum F!
        Loop
        Sum F@ SumSpan I>F F/  ;

 : ToeFilter     { | M  - }      \ Filters Toe plateau noise
                                 \ starting at NToe
        NToe @                   \ working backwards halfway to 0.
        2 / 1 +
        1
        DO
           NToe @ DUP
           I
           -  -> M
           SumAve                \  average at (M = NToe - I)
           FDUP  Bottom F!       \ becomes final value of 'bottom'
           M Temp-Arr F!         \ at exit from loop.
        LOOP
        M
        0
        DO
           Bottom F@ I Temp-Arr F!  \ Sets rest of toe plateau
                                    \  at value,  'Bottom'.
        Loop ;
```

```
: ShoulderFilter  { | M  - }    \ Filters Shoulder plateau noise
       256                     \ starting at NShoulder
       NShoulder @ -           \ working backwards halfway to 0.
       2 / 1 +  1
       DO
          NShoulder @ DUP
          I
          +  -> M
          SumAve               \ average at (M = NShoulder + I)
          FDUP  Top F!         \ becomes value of 'Top'
          M Temp-Arr F!        \ at exit from loop.
       LOOP
       256 M
       DO
          Top F@ I Temp-Arr F! \ Sets rest of toe plateau
       Loop ;                  \ at value,  'Bottom'.

: Filter ( - )                 \ Command for Shift Variant Filter
       LocateN
       -1 NFlag !
       1 Direction !
       ToeFilter
       -1 Direction !
       ShoulderFilter
       NToe @
       0
       DO
          I Temp-Arr F@
          I Data-Arr F!
       LOOP
       256
       NShoulder @ 1+
       DO
          I Temp-Arr F@
          I Data-Arr F!
       LOOP
       -1 PlotFlag !
       2 Type# ! ;

: Chop          ( - )          \ Chop filters the edge
       LocateN                 \ stored in Data-Arr.
       256 NShoulder @
       DO
          1. I Data-Arr F!
       LOOP
       NToe @ 0
       DO
          0. I Data-Arr F!
       LOOP
       1 nflag !
       1 plotflag ! ;
```

TRANSFORMING EDGE DATA TO OBTAIN OTF

```
\ ################ Tatian's Method ####################

\ The word "Mtf" Transforms the data in Data-Arr
\ and leaves the modulus in the array Mtf-Arr.

\ Algorithms perform the operations
\ described by these formulas:


\ MTF = { (OTFReal)^2 + (OTFImaginary)^2 }^1/2


\                          M
\ OTFReal = 2PiFE {summmation [e(nE)Sin (-2PiFnE)]} +
\                    n = N
\                           {Cos[(M+1/2)2PiFE]} / Sinc(FE)


\                           M
\ OTFImaginary = 2PiFE{Summation[e(nE)Cos(-2PiFnE)]} +
\                         n = N
\                           {Sin[(M+1/2)2PiFE]} / Sinc(FE)


\ ################### Source Code ##################

: SincFE   ( - Fn )              \ Returns sin(PiFE)/PiFE
       PiFE2 F@ 2. F/ FDUP       \ or 1 if PiFE is 0.
       0. F=
       IF
          FDROP
          1.
       ELSE
          FDUP FSIN FSWAP F/
       THEN ;

: SummationSin   ( - fn )        \ Sums the product of edge
       0.  Sum F!                \ function data with Sin(2PiFnE).
       ShoulderEnd @ 1 +
       ToeStart @
       DO
          I Data-Arr F@
          PiFE2 F@
          I  I>F F*
          FSin
          F*
          Sum F@ F+ Sum F!
       LOOP
       Sum F@ ;
```

```
: SummationCos    ( - fn )          \ Sums the product of edge
        0.  Sum F!                  \ function data with Cos(2piFnE).
        ShoulderEnd @ 1 +
        ToeStart @
        DO
           I Data-Arr F@
           PiFE2 F@ I I>F F*
           FCos
           F*
           Sum F@ F+ Sum F!
        LOOP
        Sum F@ ;

: CorrectionReal  ( - fn )          \ {Cos[(M+1/2)2PiFE]} / Sinc(FE)
        PiFE2 F@
        ShoulderEnd @ I>F .5 F+
        F*
        FCos
        SincFE F/ ;

: CorrectionImag  ( - fn )          \ {Sin[(M+1/2)2PiFE]} / Sinc(FE)
        PiFE2 F@
        ShoulderEnd @ I>F .5 F+
        F*
        FSin
        SincFE F/ ;

:  OTFReal        ( - fn )
        SummationSin
        PiFE2 F@ F*
        CorrectionReal
        F+ ;

: OTFImag         ( - fn )
        SummationCos
        PiFE2 F@ F*
        CorrectionImag
        F- ;

: Normalize       ( - )             \ Normalizes MTF.
        257 0
        DO
           0. I Data-Arr F!
        LOOP
        120 0
        DO
           I MTF-Arr F@ 1 MTF-Arr F@ F/
           I Data-Arr F!
        LOOP ;
```

```
: MTF    ( - )              \ Finds the MTF
         120 0              \ of data in Data-Arr.
         FreqScale
         DO
            FreqIncr F@ i I>F F*
            FDUP Frequency F!
            PiE2 F* PiFE2 F!
            OTFReal
            2. Fy^x
            OTFImag
            2. Fy^x
            F+
            FSQRT
            i MTF-Arr F!
         LOOP
         Normalize
         0 PlotFlag !
         3 Type# ! ;

: ExpMTF   ( n1 n2 - )   \ Expects center error and spread error.
         SelectPts       \ Filters selected edge with these
         Getonefile      \ parameters and finds mtf, storing this
         Filter          \ in the selected file.
         MTF
         Makefile ;

: MTfs     ( n - )       \ Consecutively grabs 20 edge files,
         FiltCase !      \ applies the filter, Chop, or no
         20 0            \ filter, finds each MTF and
         DO              \ creates a file for it.
             i 1+ . CR   \ User must select edge files
             GetOneFile  \ and name mtf files.
             FiltCase @  \ Expects the constants Filt, Chp, or NoFilt
             CASE
                1 OF Filter  i 1+  ." Filter " . ." Done " CR  ENDOF
                2 OF Chop ." Chop " i 1+ . ." Done " CR  ENDOF
                3 OF ." Unfiltered " CR  ENDOF
             ENDCASE
             Mtf
             10 CALL SYSBEEP
             Makefile
         LOOP ;
```

ANALYZING MODULATION TRANSFER FUNCTIONS

```
##########################################################################

: TransferFiles  ( a1 a2 - )     \ Transfers data from one FP array
        SWAP                     \ to another.  Expects the beginning
        256 0                    \ address of the source array followed
        DO                       \ by the beginning address of the
            i 10 * + F@          \ receiving array.
            i 10 * + F!
        LOOP ;

: ClrAbsArr      ( - )           \ Sets all items in Abs-Arr to 0.
        120 0
        DO
            0. i Abs-Arr F!
        LOOP ;


: Compare  {| nnn - Fn}  \ Returns average root mean square of the
        0 -> nnn         \ differences between data in DataII-Arr
                         \ DataI-Arr the latter being a standard.
        0. SumDif F!     \ Stops when first array reaches
        2 GetNthFile     \ a selected value, "Llimit".
        BEGIN            \ DataI-Arr must already exist.
           nnn DataI-Arr F@ FDUP
           nnn DataII-Arr F@
           F-
           2. FY^X
           SumDif F@ F+
           SumDif F!
           1 +> nnn
           Llimit F@ F<
        UNTIL
        SumDif F@
        .5 FY^X
        nnn 1 - I>F
        F/ 5 FIXED
        FDUP ." RMSDif = " F. CR ;

: MeanCalc       { n adr | - n }
                                 \ Returns mean for n points of the FP
        0. Sumx F!               \ data in the array with the beginning
        n  0                     \ address, "adr".
        DO
           adr i incr10 F@
           Sumx F@ F+ Sumx F!
        LOOP
        Sumx F@ n I>F F/ ;
```

```
: StDevCalc{ n adr | - n }      \ Returns the standard deviation of
       0. Sumdif2 F!            \ n items in the array having
       n  0                     \ the beginning address adr.
       DO                       \ Expects that the mean for that
          adr i incr10 F@       \ block of data is stored in the
          Mean F@ F-            \ variable, "mean".
          2. Fy^x
          SumDif2 F@ F+ SumDif2 F!
       LOOP
       SumDif2 F@
       n I>F 1. F-
       F/
       FSQRT ;

  Stats   (n adr -)         \ Stores the mean and standard deviation
       2DUP                 \ of n items of data in the array
       MeanCalc             \ having the beginning address "adr"
       Mean F!              \ in the variables, "Mean" and "StDev"
       StDevCalc            \  respectively.
       StDev F! ;

: CompareM  { n# | - }      \ Compares n mtf's with a standard mtf
       ClrAbsArr            \ which is located in DataI-Arr
                            \ which must exist.
       n# 0                 \ Records the results (RMS differences)
       DO                   \ for each frequency in Data-Arr
           i 1 + . CR       \ Run 1 getnthfile first to store 00 mtf
           Compare          \ standard.
           i Rms-Arr F!
       LOOP
       120 0
       DO
           i Abs-Arr F@
           n# I>F F/ 5. F*
           i Data-Arr F!
       LOOP
       n# 0 RMS-Arr
       Stats
       Mean F@
       StDev F@
       ." RMS StDev = " F.
       ." RMS Mean  = " F.
       0 plotflag !
       3 Type# ! ;
```

```
: Comp          ( n - )          \ Does n group comparisons
        1 GetNthFile             \ of mtf data with a standard
        0                        \ MTF.  Expects n on stack.
        DO                       \ Stores absolute differences
            CR                   \ in n files. Displays Rms
                                 \ differences (mean & Std Dev)
            i                    \ for each group. Makes file of
            CASE                 \ difference data.
               0 OF ." Filter " ENDOF
               1 OF ." Chop "   ENDOF
               2 OF ." Unfiltered " ENDOF
            ENDCASE
            20 CompareM
            CR ." Press RETURN to continue. "
            BEGIN
               13
               KEY
               =
            UNTIL
            MakeFile
        LOOP ;

: SingComp      ( n - )          \ Compares single MTFs with a
        1 GetNthFile             \ standard 0 noise MTF.  Will do
        0                        \ n of these storing each under
        DO                       \ a selected file name.
        CR
            1 CompareM
            CR ." Press RETURN to continue. "
            BEGIN
               13
               KEY
               =
            UNTIL
            MakeFile
        LOOP ;
```

APPENDIX V

DERIVATION OF TATIANS METHOD:

The optical transfer function is given by the equation:

$$L(f) = i2\pi f \int_{-\infty}^{\infty} l(x)\, e^{-i2\pi fx}\, dx \qquad \text{Eq AV.1}$$

Where:

L(f) = optical transfer function

f = spatial frequency

l(x) = line spread function.

Knowing that:

$$l(x) = \frac{d\, e(x)}{dx} \qquad \text{(Eq Av.2)}$$

and using the Fourier transform derivative theorem, the OTF of the system can be found by:

$$L(f) = i2\pi f \cdot E(f) \qquad \text{(Eq Av.3)}$$

Where **E(f)** is the Fourier Transform of the edge trace, **e(x).**

For sampled functions this does not usually work very well. Tatian's method involves approximating the OTF where **e(nε)** is the sampled edge trace, n is the sample number and **ε** is the sample interval. Now the OTF is:

$$L(f) = i2\pi f\varepsilon \sum_{n=-\infty}^{\infty} e(n\varepsilon)\, e^{-i2\pi f n\varepsilon} \qquad \text{Eq AV.4}$$

If the edge function is normalized then **e(nε) < N = 0**

and **e(nε) > M = 1** so the above equation can be rewritten as:

$$L(f) = i2\pi f\varepsilon \sum_{n=N}^{M} e(n\varepsilon)\, e^{-i2\pi f n\varepsilon} + i2\pi f\varepsilon \sum_{n=M+1}^{\infty} e(n\varepsilon)\, e^{-i2\pi f n\varepsilon}$$

Eq Av.5

Terms from $-\infty$ to **N-1** are almost 0 so can be eliminated from the summation. Breaking the first term into its real and imaginary parts:

$$i2\pi f\varepsilon \sum_{n=N}^{M} e(n\varepsilon)\, e^{-i2\pi f n\varepsilon} = E_R(f) + iE_I(f) \qquad \text{Eq AV.6}$$

Equation AV.5 can now be expressed in real and imaginary terms:

$$L_R(f) = E_R(f) + 2\pi f\varepsilon \sum_{n=M+1}^{\infty} \sin(-2\pi f n\varepsilon) \qquad \text{Eq AV.7}$$

$$L_I(f) = E_I(f) \quad 2\pi f\varepsilon \sum_{n=M+1}^{\infty} \cos(-2\pi f n\varepsilon) \qquad \text{Eq AV.8}$$

Using the trigonometric identities:

$$\sum_{n=M+1}^{\infty} \sin(nu) = \frac{\cos\left(\left(M + \frac{1}{2}\right)u\right)}{2\sin\left(\frac{u}{2}\right)} \qquad \text{Eq AV.9}$$

and

$$\sum_{n=M+1}^{\infty} \cos(nu) = \frac{\sin\left(\left(M + \frac{1}{2}\right)u\right)}{2\sin\left(\frac{u}{2}\right)} \qquad \text{Eq AV.10}$$

Substituting equation 9 into equation 7 get:

$$L_R(f) = E_R(f) + \frac{2\pi f\varepsilon}{2\sin(\pi f\varepsilon)} \cos\left(\left(M + \frac{1}{2}\right) 2\pi f\varepsilon\right) \qquad \text{Eq AV.11}$$

and substituting equation 10 into equation 8 get :

$$L_I(f) = E_I(f) \quad \frac{2\pi f\varepsilon}{2\sin(\pi f\varepsilon)} \sin\left(\left(M + \tfrac{1}{2}\right) 2\pi f\varepsilon\right) \qquad \text{Eq AV 12}$$

Knowing that -

$$\text{sinc}(x) = \frac{\sin(\pi x)}{(\pi x)}$$

the equations can be rewritten as :

$$L_R(f) = E_R(f) + \frac{\cos\left(\left(M + \tfrac{1}{2}\right) 2\pi f\varepsilon\right)}{\text{Sinc}(f\varepsilon)} \qquad \text{Eq AV 13}$$

and

$$L_I(f) = E_I(f) \quad \frac{\sin\left(\left(M + \tfrac{1}{2}\right) 2\pi f\varepsilon\right)}{\text{Sinc}(f\varepsilon)} \qquad \text{Eq AV.14}$$

References:

1. F. Scott, R. M. Scott, and R. Schack, "The Use of Edge Gradients in Determining Modulation Transfer Functions", Photo. Sci. Eng., 7, p. 345 -0 349, (1963)

2. B. Tatian, "Method for Obtaining the Transfer Function from the Edge Response Function", J.O.S.A., 55, p. 1014 - 1019, (1965)

3. R. Barakat, "Determination of the Optical Transfer Function Directly from the Edge Spread Function", J.O.S.A., 55, p. 1217 - 1221, (1965)

4. R. A. Jones, "An Automated Technique for Deriving MTFs from Edge Traces", Photo. Sci. Eng., 11, p. 102 - 106, (1967)

5. E. S. Blackman, "Effects of Noise on the Determination of Photographic System MTFs", Photo Sci Eng., 12, p. 244 - 250, (1968)

6. F H. Perrin, "Methods of Appraising Photographic Systems, Part I- Historical Review", J.S.M.P.T.E., 69, p. 151, (1969)

7 R. A. Jones, E. C. Yeadon, "Determination of the Spread Function from Noisy Edge Scans", Photo Sci. Eng., 13, p. 200 - 204, (1969)

8. I. Overington, M.B. Brown, "Some Simple Mathematical Methods for Derivation of Optical Image Quality Data from Edge Sharpness", S.P.I.E., Assessment of Imaging Systems, 98, p. 37 - 44, (1976)

References:

9. N. J. Schneiders and S. Bushong, "Single Step Calculation of the MTF from ERF", Medical Physics, 5, no. 1, p. 31 - 33, (1978)

10. J. Dainty and R. Shaw, Image Science, Academic Press, New York, 1974, Ch 7

11. Gaskill, J.D., Linear Systems, Fourier Transforms, and Optics, John Wiley and Sons, New York, 1978, p. 44 - 47, p. 483 - 514

12. J. E. Freund, Modern Elementary Statistics, Prentice Hall, Englewood Cliffs, New Jersey, 1979, p. 251 - 254, p. 457

13. J. C. Croteau, " Comparison of Methods of OTF/MTF Analysis for Optical Systems", Thesis, Rochester Institute of Technology, (1983)

14. J. R. Cadou, "Comparison of Two MTF Measurement Methods: Sine-wave vs Edge Gradient Analysis", Thesis, Rochester Institute of Technology, (1985)

15. D. Miley, L. Chavez, T Noyes, A. Novicov, Mach2 Forth Development System, Palo Alto Shipping Co., Menlo Park, Ca

16. E. M. Granger, Personal communication.

VITA

The author of this paper is presently working as an optical engineer with the Signal Processing group at Eastman Kodak Company developing real time wide field inspection systems.

Mr. Johnson was born February 15, 1936 in Jamestown, NY. He received a B.S degree from SUNY Fredonia and an M.S degree from SUNY Geneseo. He taught science in the Canandaigua Public Schools for 26 years before changing careers in 1986. He held the position of optical research engineer at Videk, a Kodak machine vision company before moving to Kodak in 1987. He is co-inventor of a wide field, white light, optical signal processing device presently in use at Kodak.

He now lives with his family, Jacqueline, Joel, Matthew and Kelly Johnson in a rural setting in the hills above Canandaigua Lake.